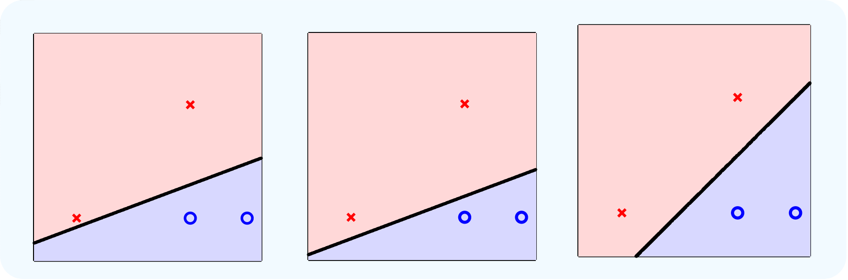
e-Chapter 8

Support Vector Machines

Linear models are powerful. The nonlinear transform and the neural network (a cascade of linear models) are tools that increase their expressive power. Increasing the expressive power has a price: overfitting and computation time. Can we get the expressive power without paying the price? The answer is yes. Our new model, the Support Vector Machine (SVM), uses a 'safety cushion' when separating the data. As a result, SVM is more robust to noise; this helps combat overfitting. In addition, SVM can work seamlessly with a new powerful tool called the *kernel*: a computationally efficient way to use high (even infinite!) dimensional nonlinear transforms. When we combine the safety cushion of SVM with the 'kernel trick', the result is an efficient powerful nonlinear model with automatic regularization. The SVM model is popular because it performs well in practice and is easy to use. This chapter presents the mathematics and algorithms that make SVM work.

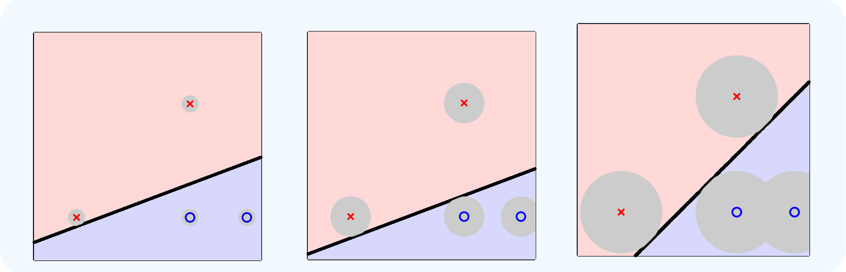
8.1 The Optimal Hyperplane

Let us revisit the perceptron model from Chapter 1. The illustrations below on a toy data set should help jog your memory. In 2 dimensions, a perceptron attempts to separate the data with a line, which is possible in this example.



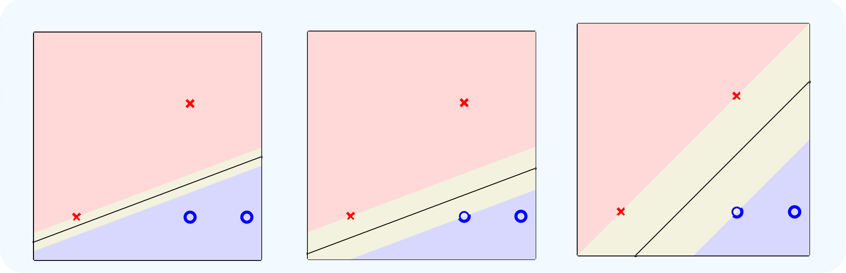
As you can see, many lines separate the data and the Perceptron Learning Algorithm (PLA) finds one of them. Do we care about which one PLA finds? All separators have $E_{\text{in}} = 0$, so the VC analysis in Chapter 2 gives the same E_{out}-bound for every separator. Well, the VC bound may say one thing, but surely our intuition says that the rightmost separator is preferred ☺.

Let's try to pin down an argument that supports our intuition. In practice, there are measurement errors – *noise*. Place identical shaded regions around each data point, with the radius of the region being the amount of possible measurement error. The true data point can lie anywhere within this 'region of uncertainty' on account of the measurement error. A separator is 'safe' with respect to the measurement error if it classifies the *true* data points correctly. That is, no matter where in its region of uncertainty the true data point lies, it is still on the correct side of the separator. The figure below shows the largest measurement errors which are safe for each separator.



A separator that can tolerate more measurement error is safer. The rightmost separator tolerates the largest error, whereas for the leftmost separator, even a small error in some data points could result in a misclassification. In Chapter 4, we saw that noise (for example measurement error) is the main cause of overfitting. Regularization helps us combat noise and avoid overfitting. In our example, the rightmost separator is more robust to noise without compromising E_{in}; it is better 'regularized'. Our intuition is well justified.

We can also quantify noise tolerance from the viewpoint of the separator. Place a cushion on each side of the separator. We call such a separator with a cushion *fat*, and we say that it separates the data if no data point lies within its cushion. Here is the largest cushion we can place around each of our three candidate separators.



To get the thickest cushion, keep extending the cushion equally on both sides of the separator until you hit a data point. The thickness reflects the amount of noise the separator can tolerate. If any data point is perturbed by at most the thickness of either side of the cushion, it will still be on the correct side of the separator. The maximum thickness (noise tolerance) possible for a separator is called its *margin*. Our intuition tells us to pick the fattest separator possible, the one with maximum margin. In this section, we will address three important questions:

1. Can we efficiently find the fattest separator?
2. Why is a fat separator better than a thin one?
3. What should we do if the data is not separable?

The first question relates to the algorithm; the second relates to E_{out}; and, the third relates to E_{in} (we will also elaborate on this question in Section 8.4).

Our discussion was in 2 dimensions. In higher dimensions, the separator is a hyperplane and our intuition still holds. Here is a warm-up.

Exercise 8.1

Assume $\mathcal{D}$ contains two data points $(\mathbf{x}_+, +1)$ and $(\mathbf{x}_-, -1)$. Show that:

(a) No hyperplane can tolerate noise radius greater than $\frac{1}{2}\|\mathbf{x}_+ - \mathbf{x}_-\|$.

(b) There is a hyperplane that tolerates a noise radius $\frac{1}{2}\|\mathbf{x}_+ - \mathbf{x}_-\|$.

8.1.1 Finding the Fattest Separating Hyperplane

Before we proceed to the mathematics, we fix a convention that we will use throughout the chapter. Recall that a hyperplane is defined by its weights $\mathbf{w}$.

Isolating the bias. We explicitly split the weight vector $\mathbf{w}$ as follows. The bias weight w_0 is taken out, and the rest of the weights $w_1, \ldots, w_d$ remain in $\mathbf{w}$. The reason is that the mathematics will treat these two types of weights differently. To avoid confusion, we will relabel w_0 to b (for bias), but continue to use $\mathbf{w}$ for $(w_1, \ldots, w_d)$.

Previous Chapters

$\mathbf{x} \in \{1\} \times \mathbb{R}^d;\ \mathbf{w} \in \mathbb{R}^{d+1}$

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}; \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}.$$

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

This Chapter

$\mathbf{x} \in \mathbb{R}^d;\ b \in \mathbb{R},\ \mathbf{w} \in \mathbb{R}^d$

$$b = \text{bias}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}; \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}.$$

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$$

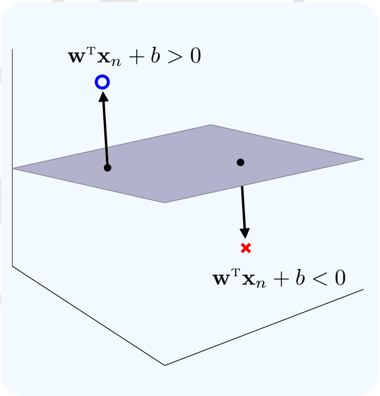
A maximum-margin separating hyperplane has two defining properties.

1. It separates the data.
2. It has the thickest cushion among hyperplanes that separate the data.

To find a separating hyperplane with maximum margin, we first re-examine the definition of a *separating* hyperplane and reshape the definition into an equivalent, more convenient one. Then, we discuss how to compute the margin of any given separating hyperplane (so that we can find the one with maximum margin). As we observed in our earlier intuitive discussion, the margin is obtained by extending the cushion until you hit a data point. That is, the margin is the distance from the hyperplane to the *nearest* data point. We thus need to become familiar with the geometry of hyperplanes; in particular, how to compute the distance from a data point to the hyperplane.

Separating hyperplanes. The hyperplane h, defined by $(b, \mathbf{w})$, separates the data if and only if for $n = 1, \ldots, N$,

$$y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) > 0. \qquad (8.1)$$



The signal $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)$ is positive for each data point. However, the magnitude of the signal is not meaningful by itself since we can make it arbitrarily small or large *for the same hyperplane* by rescaling the weights and the bias. This is because $(b, \mathbf{w})$ is the *same* hyperplane as $(b/\rho, \mathbf{w}/\rho)$ for any $\rho > 0$. By rescaling the weights, we can control the size of the signal for our data points. Let us pick a particular value of ρ,

$$\rho = \min_{n=1,\ldots,N} \; y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b),$$

which is positive because of (8.1). Now, rescale the weights to obtain the *same* hyperplane $(b/\rho, \mathbf{w}/\rho)$. For these rescaled weights,

$$\min_{n=1,\ldots,N} y_n\left(\frac{\mathbf{w}}{\rho}^{\mathrm{T}}\mathbf{x}_n + \frac{b}{\rho}\right) = \frac{1}{\rho}\min_{n=1,\ldots,N} \; y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) = \frac{\rho}{\rho} = 1.$$

Thus, for any separating hyperplane, it is always possible to choose weights so that all the signals $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)$ are of magnitude greater than or equal to 1, with equality satisfied by at least one $(\mathbf{x}_n, y_n)$. This motivates our new definition of a separating hyperplane.

Definition 8.1 (Separating Hyperplane)**.** The hyperplane h separates the data if and only if it can be represented by weights $(b, \mathbf{w})$ that satisfy

$$\min_{n=1,\ldots,N} \; y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) = 1. \qquad (8.2)$$

The conditions (8.1) and (8.2) are equivalent. *Every* separating hyperplane can be accommodated under Definition 8.1. All we did is constrain the way we *algebraically* represent such a hyperplane by choosing a (data dependent) normalization for the weights, to ensure that the magnitude of the signal is meaningful. Our normalization in (8.2) will be particularly convenient for deriving the algorithm to find the maximum-margin separator. The next exercise gives a concrete example of re-normalizing the weights to satisfy (8.2).

Exercise 8.2

Consider the data below and a 'hyperplane' $(b, \mathbf{w})$ that separates the data.

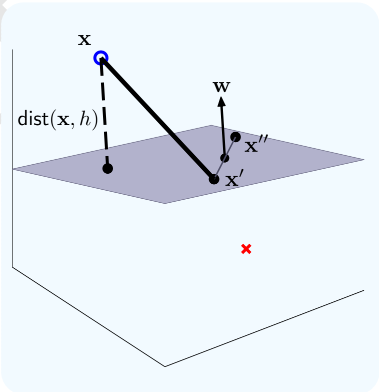
$$\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 2 & 0 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 1.2 \\ -3.2 \end{bmatrix} \quad b = -0.5$$

(a) Compute $\rho = \min_{n=1,\dots,N} y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)$.

(b) Compute the weights $\frac{1}{\rho}(b, \mathbf{w})$ and show that they satisfy (8.2).

(c) Plot both hyperplanes to show that they are the *same* separator.

Margin of a hyperplane. To compute the margin of a separating hyperplane, we need to compute the distance from the hyperplane to the *nearest* data point. As a start, let us compute the distance from an arbitrary point $\mathbf{x}$ to a separating hyperplane $h = (b, \mathbf{w})$ that satisfies (8.2). Denote this distance by $\text{dist}(\mathbf{x}, h)$. Referring to the figure on the right, $\text{dist}(\mathbf{x}, h)$ is the length of the perpendicular from $\mathbf{x}$ to h. Let $\mathbf{x}'$ be any point on the hyperplane, which means $\mathbf{w}^{\mathrm{T}}\mathbf{x}' + b = 0$. Let $\mathbf{u}$ be a unit vector that is normal to the hyperplane h. Then, $\text{dist}(\mathbf{x}, h) = |\mathbf{u}^{\mathrm{T}}(\mathbf{x} - \mathbf{x}')|$, the projection of the vector $(\mathbf{x} - \mathbf{x}')$ onto $\mathbf{u}$. We now argue that $\mathbf{w}$ is normal to the hyperplane, and so we can take $\mathbf{u} = \mathbf{w}/\|\mathbf{w}\|$. Indeed, any vector lying on the hyperplane can be expressed by $(\mathbf{x}'' - \mathbf{x}')$ for some $\mathbf{x}', \mathbf{x}''$ on the hyperplane, as shown. Then, using $\mathbf{w}^{\mathrm{T}}\mathbf{x} = -b$ for points on the hyperplane,



$$\mathbf{w}^{\mathrm{T}}(\mathbf{x}'' - \mathbf{x}') = \mathbf{w}^{\mathrm{T}}\mathbf{x}'' - \mathbf{w}^{\mathrm{T}}\mathbf{x}' = -b + b = 0.$$

Therefore, $\mathbf{w}$ is orthogonal to *every* vector in the hyperplane, hence it is the normal vector as claimed. Setting $\mathbf{u} = \mathbf{w}/\|\mathbf{w}\|$, the distance from $\mathbf{x}$ to h is

$$\text{dist}(\mathbf{x}, h) = |\mathbf{u}^{\mathrm{T}}(\mathbf{x} - \mathbf{x}')| = \frac{|\mathbf{w}^{\mathrm{T}}\mathbf{x} - \mathbf{w}^{\mathrm{T}}\mathbf{x}'|}{\|\mathbf{w}\|} = \frac{|\mathbf{w}^{\mathrm{T}}\mathbf{x} + b|}{\|\mathbf{w}\|},$$

where we used $\mathbf{w}^{\mathrm{T}}\mathbf{x}' = -b$ in the last step, since $\mathbf{x}'$ is on h. You can now see why we separated the bias b from the weights $\mathbf{w}$: the distance calculation treats these two parameters differently.

We are now ready to compute the margin of a separating hyperplane. Consider the data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and hyperplane $h = (b, \mathbf{w})$ that satisfies the separating condition (8.2). Since $y_n = \pm 1$,

$$|\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b| = |y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)| = y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b),$$

where the last equality follows because $(b, \mathbf{w})$ separates the data, which implies that $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)$ is positive. So, the distance of a data point to h is

$$\mathsf{dist}(\mathbf{x}_n, h) = \frac{y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)}{\|\mathbf{w}\|}.$$

Therefore, the data point that is nearest to the hyperplane has distance

$$\min_{n=1,\ldots,N} \mathsf{dist}(\mathbf{x}_n, h) = \frac{1}{\|\mathbf{w}\|} \cdot \min_{n=1,\ldots,N} y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) = \frac{1}{\|\mathbf{w}\|},$$

where the last equality follows because $(b, \mathbf{w})$ separates the data and satisfies (8.2). This simple expression for the distance of the nearest data point to the hyperplane is the entire reason why we chose to normalize $(b, \mathbf{w})$ as we did, by requiring (8.2).[1] For any separating hyperplane satisfying (8.2), the margin is $1/\|\mathbf{w}\|$. If you hold on a little longer, you are about to reap the full benefit, namely a simple algorithm for finding the optimal (fattest) hyperplane.

The fattest separating hyperplane. The maximum-margin separating hyperplane $(b^*, \mathbf{w}^*)$ is the one that satisfies the separation condition (8.2) with minimum weight-norm (since the margin is the inverse of the weight-norm). Instead of minimizing the weight-norm, we can equivalently minimize $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$, which is analytically more friendly. Therefore, to find this optimal hyperplane, we need to solve the following optimization problem.

$$\begin{aligned} \underset{b,\mathbf{w}}{\text{minimize:}} \quad & \tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \\ \text{subject to:} \quad & \min_{n=1,\ldots,N} y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) = 1. \end{aligned} \tag{8.3}$$

The constraint ensures that the hyperplane separates the data as per (8.2). Observe that the bias b does not appear in the quantity being minimized, but it is involved in the constraint (again, b is treated differently than $\mathbf{w}$). To make the optimization problem easier to solve, we can replace the single constraint

[1] Some terminology: the parameters $(b, \mathbf{w})$ of a separating hyperplane that satisfy (8.2) are called the canonical representation of the hyperplane. For a separating hyperplane in its canonical representation, the margin is just the inverse norm of the weights.

$\min_n y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) = 1$ with N 'looser' constraints $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1$ for $n = 1, \ldots, N$ and solve the optimization problem:

$$\begin{aligned} &\underset{b,\mathbf{w}}{\text{minimize:}} && \tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} && \\ &\text{subject to:} && y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 && (n = 1, \cdots, N). \end{aligned} \tag{8.4}$$

The constraint in (8.3) implies the constraints in (8.4), which means that the constraints in (8.4) are looser. Fortunately, *at the optimal solution*, the constraints in (8.4) become equivalent to the constraint in (8.3) as long as there are both positive and negative examples in the data. After solving (8.4), we will show that the constraint of (8.3) is automatically satisfied. This means that we will also have solved (8.3).

To do that, we will use a proof by contradiction. Suppose that the solution $(b^*, \mathbf{w}^*)$ of (8.4) has

$$\rho^* = \min_n y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) > 1,$$

and therefore is not a solution to (8.3). Consider the rescaled hyperplane $(b, \mathbf{w}) = \frac{1}{\rho^*}(b^*, \mathbf{w}^*)$, which satisfies the constraints in (8.4) by construction. For $(b, \mathbf{w})$, we have that $\|\mathbf{w}\| = \frac{1}{\rho^*}\|\mathbf{w}^*\| < \|\mathbf{w}^*\|$ (unless $\mathbf{w}^* = \mathbf{0}$), which means that $\mathbf{w}^*$ cannot be optimal for (8.4) unless $\mathbf{w}^* = \mathbf{0}$. It is not possible to have $\mathbf{w}^* = \mathbf{0}$ since this would not correctly classify both the positive and negative examples in the data.

We will refer to this fattest separating hyperplane as *the optimal hyperplane*. To get the optimal hyperplane, all we have to do is solve the optimization problem in (8.4).

Example 8.2. The best way to get a handle on what is going on is to carefully work through an example to see how solving the optimization problem in (8.4) results in the optimal hyperplane $(b^*, \mathbf{w}^*)$. In two dimensions, a hyperplane is specified by the parameters (b, w_1, w_2). Let us consider the toy data set that was the basis for the figures on page 8-2. The data matrix and target values, together with the separability constraints from (8.4) are summarized below. The inequality on a particular row is the separability constraint for the corresponding data point in that row.

$$\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 2 & 0 \\ 3 & 0 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix} \qquad \begin{aligned} -b &\geq 1 && \text{(i)} \\ -(2w_1 + 2w_2 + b) &\geq 1 && \text{(ii)} \\ 2w_1 + b &\geq 1 && \text{(iii)} \\ 3w_1 + b &\geq 1 && \text{(iv)} \end{aligned}$$

Combining (i) and (iii) gives

$$w_1 \geq 1.$$

Combining (ii) and (iii) gives

$$w_2 \leq -1.$$

This means that $\frac{1}{2}(w_1^2 + w_2^2) \geq 1$ with equality when $w_1 = 1$ and $w_2 = -1$. One can easily verify that

$$(b^* = -1, w_1^* = 1, w_2^* = -1)$$

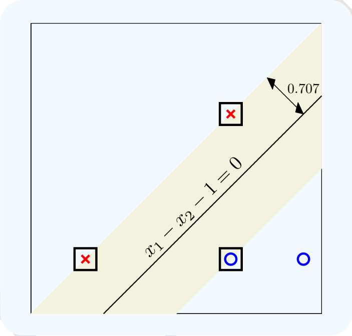
satisfies all four constraints, minimizes $\frac{1}{2}(w_1^2 + w_2^2)$, and therefore gives the optimal hyperplane. The optimal hyperplane is shown in the following figure.

<u>Optimal Hyperplane</u>

$$g(\mathbf{x}) = \text{sign}(x_1 - x_2 - 1)$$

$$\text{margin:} \quad \frac{1}{\|\mathbf{w}^*\|} = \frac{1}{\sqrt{2}} \approx 0.707.$$

Data points (i), (ii) and (iii) are boxed because their separation constraints are exactly met: $y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) = 1$.



For data points which meet their constraints exactly, $\text{dist}(\mathbf{x}_n, g) = \frac{1}{\|\mathbf{w}^*\|}$. These data points sit on the boundary of the cushion and play an important role. They are called *support vectors*. In a sense, the support vectors are 'supporting' the cushion and preventing it from expanding further. □

Exercise 8.3

For separable data that contain both positive and negative examples, and a separating hyperplane h, define the positive-side margin $\rho_+(h)$ to be the distance between h and the nearest data point of class $+1$. Similarly, define the negative-side margin $\rho_-(h)$ to be the distance between h and the nearest data point of class -1. Argue that if h is the optimal hyperplane, then $\rho_+(h) = \rho_-(h)$. That is, the thickness of the cushion on either side of the optimal h is equal.

We make an important observation that will be useful later. In Example 8.2, what happens to the optimal hyperplane if we removed data point (iv), the non-support vector? Nothing! The hyperplane remains a separator with the *same* margin. Even though we removed a data point, a larger margin cannot be achieved since all the support vectors that previously prevented the margin from expanding are still in the data. So the hyperplane remains optimal. Indeed, to compute the optimal hyperplane, only the support vectors are needed; the other data could be thrown away.

Quadratic Programming (QP). For bigger data sets, manually solving the optimization problem in (8.4) as we did in Example 8.2 is no longer feasible.

The good news is that (8.4) belongs to a well-studied family of optimization problems known as quadratic programming (QP). Whenever you minimize a (convex) *quadratic* function, subject to *linear* inequality constraints, you can use quadratic programming. Quadratic programming is such a well studied area that excellent, publicly available solvers exist for many numerical computing platforms. We will not need to know how to solve a quadratic programming problem; we will only need to know how to take any given problem and convert it into a standard form which is then input to a QP-solver. We begin by describing the standard form of a QP-problem:

$$\begin{aligned} \underset{\mathbf{u}\in\mathbb{R}^L}{\text{minimize:}} \quad & \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} \\ \text{subject to:} \quad & \mathbf{a}_m^{\mathrm{T}}\mathbf{u} \ge c_m \qquad (m = 1, \cdots, M). \end{aligned} \tag{8.5}$$

The variables to be optimized are the components of the vector $\mathbf{u}$ which has dimension L. All the other parameters Q, $\mathbf{p}$, $\mathbf{a}_m$ and c_m for $m = 1, \ldots, M$ are user-specified. The quantity being minimized contains a quadratic term $\frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u}$ and a linear term $\mathbf{p}^{\mathrm{T}}\mathbf{u}$. The coefficients of the quadratic terms are in the $L \times L$ matrix Q: $\frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} = \sum_{i=1}^{L}\sum_{j=1}^{L} q_{ij}u_iu_j$. The coefficients of the linear terms are in the $L \times 1$ vector $\mathbf{p}$: $\mathbf{p}^{\mathrm{T}}\mathbf{u} = \sum_{i=1}^{L} p_iu_i$. There are M *linear* inequality constraints, each one being specified by an $L \times 1$ vector $\mathbf{a}_m$ and scalar c_m. For the QP-problem to be convex, the matrix Q needs to be positive semi-definite. It is more convenient to specify the $\mathbf{a}_m$ as rows of an $M \times L$ matrix A and the c_m as components of an $M \times 1$ vector $\mathbf{c}$:

$$\mathrm{A} = \begin{bmatrix} \text{—}\mathbf{a}_1^{\mathrm{T}}\text{—} \\ \vdots \\ \text{—}\mathbf{a}_M^{\mathrm{T}}\text{—} \end{bmatrix} \qquad \mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_M \end{bmatrix}.$$

Using the matrix representation, the QP-problem in (8.5) is written as

$$\begin{aligned} \underset{\mathbf{u}\in\mathbb{R}^L}{\text{minimize:}} \quad & \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} \\ \text{subject to:} \quad & \mathrm{A}\mathbf{u} \ge \mathbf{c}. \end{aligned} \tag{8.6}$$

The M inequality constraints are all contained in the single vector constraint $\mathrm{A}\mathbf{u} \ge \mathbf{c}$ (which must hold for each component). We will write

$$\mathbf{u}^* \leftarrow \mathsf{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$$

to denote the process of running a QP-solver on the input $(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$ to get an optimal solution $\mathbf{u}^*$ for (8.6).[2]

[2] A quick comment about the input to QP-solvers. Some QP-solvers take the reverse inequality constraints $\mathrm{A}\mathbf{u} \le \mathbf{c}$, which simply means you need to negate A and $\mathbf{c}$ in (8.6). An equality constraint is two inequality constraints ($a = b \iff a \ge b$ and $a \le b$). Some solvers will accept separate upper and lower bound constraints on each component of $\mathbf{u}$. All these different variants of quadratic programming can be represented by the standard problem with linear inequality constraints in (8.6). However, special types of constraints like equality and upper/lower bound constraints can often be handled in a more numerically stable way than the general linear inequality constraint.

We now show that our optimization problem in (8.4) is indeed a QP-problem. To do so, we must identify Q, $\mathbf{p}$, A and $\mathbf{c}$. First, observe that the quantities that need to be solved for (the optimization variables) are $(b, \mathbf{w})$; collecting these into $\mathbf{u}$, we identify the optimization variable $\mathbf{u} = \left[\begin{smallmatrix} b \\ \mathbf{w} \end{smallmatrix}\right] \in \mathbb{R}^{d+1}$. The dimension of the optimization problem is $L = d + 1$. The quantity we are minimizing is $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$. We need to write this in the form $\frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u}$. Observe that

$$\mathbf{w}^{\mathrm{T}}\mathbf{w} = \begin{bmatrix} b & \mathbf{w}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} 0 & \mathbf{0}_d^{\mathrm{T}} \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix} \begin{bmatrix} b \\ \mathbf{w}^{\mathrm{T}} \end{bmatrix} = \mathbf{u}^{\mathrm{T}} \begin{bmatrix} 0 & \mathbf{0}_d^{\mathrm{T}} \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix} \mathbf{u},$$

where I_d is the $d \times d$ identity matrix and $\mathbf{0}_d$ the d-dimensional zero vector. We can identify the quadratic term with $\mathrm{Q} = \left[\begin{smallmatrix} 0 & \mathbf{0}_d^{\mathrm{T}} \\ \mathbf{0}_d & \mathrm{I}_d \end{smallmatrix}\right]$, and there is no linear term, so $\mathbf{p} = \mathbf{0}_{d+1}$. As for the constraints in (8.4), there are N of them in (8.4), so $M = N$. The n-th constraint is $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1$, which is equivalent to

$$\begin{bmatrix} y_n & y_n\mathbf{x}_n^{\mathrm{T}} \end{bmatrix} \mathbf{u} \geq 1,$$

and that corresponds to setting $\mathbf{a}_n^{\mathrm{T}} = y_n \begin{bmatrix} 1 & \mathbf{x}_n^{\mathrm{T}} \end{bmatrix}$ and $c_n = 1$ in (8.5). So, the matrix A contains rows which are very related to an old friend from Chapter 3, the data matrix X augmented with a column of 1s. In fact, the n-th row of A is just the n-th row of the data matrix but multiplied by its label y_n. The constraint vector $\mathbf{c} = \mathbf{1}_N$, an N-dimensional vector of ones.

Exercise 8.4

Let Y be an $N \times N$ diagonal matrix with diagonal entries $\mathrm{Y}_{nn} = y_n$ (a matrix version of the target vector $\mathbf{y}$). Let X be the data matrix augmented with a column of 1s. Show that

$$\mathrm{A} = \mathrm{YX}.$$

We summarize below the algorithm to get an optimal hyperplane, which reduces to a QP-problem in $d + 1$ variables with N constraints.

Linear Hard-Margin SVM with QP

1: Let $\mathbf{p} = \mathbf{0}_{d+1}$ ($(d+1)$-dimensional zero vector) and $\mathbf{c} = \mathbf{1}_N$ (N-dimensional vector of ones). Construct matrices Q and A, where

$$\mathrm{Q} = \begin{bmatrix} 0 & \mathbf{0}_d^{\mathrm{T}} \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix}, \qquad \mathrm{A} = \underbrace{\begin{bmatrix} y_1 & \text{—}y_1\mathbf{x}_1^{\mathrm{T}}\text{—} \\ \vdots & \vdots \\ y_N & \text{—}y_N\mathbf{x}_N^{\mathrm{T}}\text{—} \end{bmatrix}}_{\text{signed data matrix}}.$$

2: Calculate $\begin{bmatrix} b^* \\ \mathbf{w}^* \end{bmatrix} = \mathbf{u}^* \leftarrow \mathsf{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$.

3: Return the hypothesis $g(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^{*\mathrm{T}}\mathbf{x} + b^*)$.

The learning model we have been discussing is known as the *linear hard-margin support vector machine* (linear hard-margin SVM). Yes, it sounds like something out of a science fiction book ☺. Don't worry, it is just a name that describes an algorithm for constructing an optimal linear model, and in this algorithm only a few of the data points have a role to play in determining the final hypothesis - these few data points are called *support vectors*, as mentioned earlier. The margin being hard means that no data is allowed to lie inside the cushion. We can relax this condition, and we will do so later in this chapter.

Exercise 8.5

Show that the matrix Q described in the linear hard-margin SVM algorithm above is positive semi-definite (that is $\mathbf{u}^{\mathsf{T}}\mathrm{Q}\mathbf{u} \geq 0$ for any $\mathbf{u}$).

The result means that the QP-problem is convex. Convexity is useful because this makes it 'easy' to find an optimal solution. In fact, standard QP-solvers can solve our convex QP-problem in $O((N+d)^3)$.

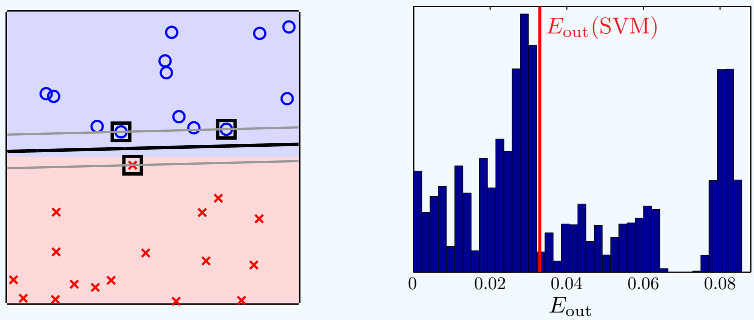
Example 8.3. We can explicitly construct the QP-problem for our toy example in Example 8.2. We construct Q, $\mathbf{p}$, A, and $\mathbf{c}$ as follows.

$$\mathrm{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathrm{A} = \begin{bmatrix} -1 & 0 & 0 \\ -1 & -2 & -2 \\ 1 & 2 & 0 \\ 1 & 3 & 0 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

A standard QP-solver gives $(b^*, w_1^*, w_2^*) = (-1, 1, -1)$, the same solution we computed manually, but obtained in less than a millisecond. □

In addition to standard QP-solvers that are publicly available, there are also specifically tailored solvers for the linear SVM, which are often more efficient for learning from *large-scale* data, characterized by a large N or d. Some packages are based on a version of stochastic gradient descent (SGD), a technique for optimization that we introduced in Chapter 3, and other packages use more sophisticated optimization techniques that take advantage of special properties of SVM (such as the redundancy of non-support vectors).

Example 8.4 (Comparison of SVM with PLA)**.** We construct a toy data set and use it to compare SVM with the perceptron learning algorithm. To generate the data, we randomly generate $x_1 \in [0, 1]$ and $x_2 \in [-1, 1]$ with the target function being +1 above the x_1-axis; $f(\mathbf{x}) = \text{sign}(x_2)$. In this experiment, we used the version of PLA that updates the weights using the misclassified point $\mathbf{x}_n$ with lowest index n. The histogram in Figure 8.1 shows what will typically happen with PLA. Depending on the ordering of the data, PLA can sometimes get lucky and beat the SVM, and sometimes it can be much worse. The SVM (maximum-margin) classifier does not depend on the (random) ordering of the data. □



(a) Data and SVM separator (b) Histogram of E_{out}(PLA)

Figure 8.1: (a) The SVM classifier from data generated using $f(\mathbf{x}) = \text{sign}(x_2)$ (the blue region is $f(\mathbf{x}) = +1$). The margin (cushion) is inside the gray lines and the three support vectors are enclosed in boxes. (b) A histogram of E_{out} for PLA classifiers using random orderings of the data.

Exercise 8.6

Construct a toy data set with $N = 20$ using the method in Example 8.4.

(a) Run the SVM algorithm to obtain the maximum margin separator $(b, \mathbf{w})_{\text{SVM}}$ and compute its E_{out} and margin.

(b) Construct an ordering of the data points that results in a hyperplane with bad E_{out} when PLA is run on it. *[Hint: Identify a positive and negative data point for which the perpendicular bisector separating these two points is a bad separator. Where should these two points be in the ordering? How many iterations will PLA take?]*

(c) Create a plot of your two separators arising from SVM and PLA.

8.1.2 Is a Fat Separator Better?

The SVM (optimal hyperplane) is a linear model, so it cannot fit certain simple functions as discussed in Chapter 3. But, on the positive side, the SVM also inherits the good generalization capability of the simple linear model since the VC dimension is bounded by $d + 1$. Does the support vector machine gain any more generalization ability by maximizing the margin, providing a safety cushion so to speak? Now that we have the algorithm that gets us the optimal hyperplane, we are in a position to shed some light on this question.

We already argued intuitively that the optimal hyperplane is robust to noise. We now show this link to regularization more explicitly. We have seen an optimization problem similar to the one in (8.4) before. Recall the soft-

order constraint when we discussed regularization in Chapter 4,

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize:}} \quad & E_{\text{in}}(\mathbf{w}) \\ \text{subject to:} \quad & \mathbf{w}^{\mathrm{T}}\mathbf{w} \leq C. \end{aligned}$$

Ultimately, this led to weight-decay regularization. In regularization, we minimize E_{in} given a budget C for $\mathbf{w}^{\mathrm{T}}\mathbf{w}$. For the optimal hyperplane (SVM), we minimize $\mathbf{w}^{\mathrm{T}}\mathbf{w}$ subject to a budget on E_{in} (namely $E_{\text{in}} = 0$). In a sense, the optimal hyperplane is doing automatic weight-decay regularization.

	optimal hyperplane	regularization
minimize:	$\mathbf{w}^{\mathrm{T}}\mathbf{w}$	E_{in}
subject to:	$E_{\text{in}} = 0$	$\mathbf{w}^{\mathrm{T}}\mathbf{w} \leq C$

Both methods are trying to fit the data with small weights. That is the gist of regularization. This link to regularization is interesting, but does it help? Yes, both in theory and practice. We hope to convince you of three things.

(i) A larger-margin separator yields better performance in practice. We will illustrate this with a simple empirical experiment.

(ii) Fat hyperplanes generalize better than thin hyperplanes. We will show this by bounding the number of points that fat hyperplanes can shatter. Our bound is less than $d + 1$ for fat enough hyperplanes.

(iii) The out-of-sample error can be small, even if the dimension d is large. To show this, we bound the cross validation error E_{cv} (recall that E_{cv} is a proxy for E_{out}). Our bound does not explicitly depend on the dimension.

(i) Larger Margin is Better. We will use the toy learning problem in Example 8.4 to study how separators with different margins perform. We randomly generate a separable data set of size $N = 20$. There are infinitely many separating hyperplanes. We sample these separating hyperplanes randomly.

For each random separating hyperplane h, we can compute $E_{\text{out}}(h)$ and the margin $\rho(h)/\rho(\text{SVM})$ (normalized by the maximum possible margin achieved by the SVM). We can now plot how $E_{\text{out}}(h)$ depends on the *fraction* of the available margin that h uses. For each data set, we sample 50,000 random separating hyperplanes and then average the results over several thousands of data sets. Figure 8.2 shows the dependence of E_{out} versus margin.

Figure 8.2 clearly suggests that when choosing a random separating hyperplane, it is better to choose one with larger margin. The SVM, which picks *the* hyperplane with largest margin, is doing a much better job than one of the (typical) random separating hyperplanes. Notice that once you get to large enough margin, increasing the margin further can hurt E_{out}: it is possible to slightly improve the performance among random hyperplanes by slightly sacrificing on the maximum margin and perhaps improving in other ways. Let's now build some theoretical evidence for why large margin separation is good.

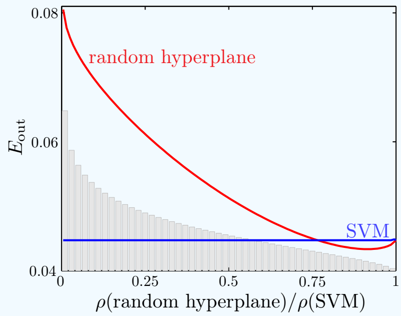


Figure 8.2: Dependence of E_{out} versus the margin ρ for a random separating hyperplane. The shaded histogram in the background shows the relative frequencies of a particular margin when selecting a random hyperplane.

(ii) Fat Hyperplanes Shatter Fewer Points. The VC dimension of a hypothesis set is the maximum number of points that can be shattered. A smaller VC dimension gives a smaller generalization error bar that links E_{in} to E_{out} (see Chapter 2 for details). Consider the hypothesis set $\mathcal{H}_\rho$ containing *all* fat hyperplanes of width (margin) at least ρ. A dichotomy on a data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ can be implemented by a hypothesis $h \in \mathcal{H}_\rho$ if $y_n = h(\mathbf{x}_n)$ and none of the $\mathbf{x}_n$ lie inside the margin of h. Let $d_{\text{VC}}(\rho)$ be the maximum number of points that $\mathcal{H}_\rho$ can shatter.[3] Our goal is to show that restricting the hypothesis set to fat separators can decrease the number of points that can be shattered, that is, $d_{\text{VC}}(\rho) < d + 1$. Here is an example.

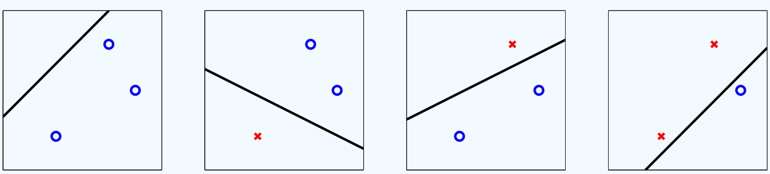


Figure 8.3: Thin hyperplanes can implement all 8 dichotomies (the other 4 dichotomies are obtained by negating the weights and bias).

Thin (zero thickness) separators can shatter the three points shown above. As we increase the thickness of the separator, we will soon not be able to implement the rightmost dichotomy. Eventually, as we increase the thickness even

[3]Technically a hypothesis in $\mathcal{H}_\rho$ is not a classifier because inside its margin the output is not defined. Nevertheless we can still compute the maximum number of points that can be shattered and this 'VC dimension' plays a similar role to d_{VC} in establishing a generalization error bar for the model, albeit using more sophisticated analysis.

more, the only dichotomies we can implement will be the constant dichotomies.

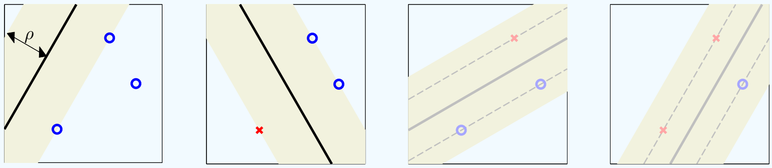


Figure 8.4: Only 4 of the 8 dichotomies can be separated by hyperplanes with thickness ρ. The dashed lines show the thickest possible separator for each non-separable dichotomy.

In Figure 8.4, there is no ρ-thick hyperplane that can separate the right two dichotomies. This example illustrates why thick hyperplanes implement fewer of the possible dichotomies. Note that the hyperplanes only 'look' thick because the data are close together. If we moved the data further apart, then even a thick hyperplane can implement all the dichotomies. What matters is the thickness of the hyperplane *relative* to the spacing of the data.

Exercise 8.7

Assume that the data is restricted to lie in a unit sphere.

(a) Show that $d_{\text{VC}}(\rho)$ is non-increasing in ρ.

(b) In 2 dimensions, show that $d_{\text{VC}}(\rho) < 3$ for $\rho > \frac{\sqrt{3}}{2}$. *[Hint: Show that for any 3 points in the unit disc, there must be two that are within distance $\sqrt{3}$ of each other. Use this fact to construct a dichotomy that cannot be implemented by any ρ-thick separator.]*

The exercise shows that for a bounded input space, thick separators can shatter fewer than $d+1$ points. In general, we can prove the following result.

Theorem 8.5 (VC dimension of Fat Hyperplanes)**.** Suppose the input space is the ball of radius R in $\mathbb{R}^d$, so $\|\mathbf{x}\| \leq R$. Then,

$$d_{\text{VC}}(\rho) \leq \left\lceil R^2/\rho^2 \right\rceil + 1,$$

where $\left\lceil R^2/\rho^2 \right\rceil$ is the smallest integer greater than or equal to R^2/ρ^2.

We also know that the VC dimension is at most $d+1$, so we can pick the better of the two bounds, and we gain when R/ρ is small. The most important fact about this margin-based bound is that it does not explicitly depend on the dimension d. This means that if we transform the data to a high, even infinite, dimensional space, as long as we use fat enough separators, we obtain

good generalization. Since this result establishes a crucial link between the margin and good generalization, we give a simple, though somewhat technical, proof.

Begin safe skip: The proof is cute but not essential. A similar green box will tell you when to rejoin.

Proof. Fix $\mathbf{x}_1, \ldots, \mathbf{x}_N$ that are shattered by hyperplanes with margin ρ. We will show that when N is even, $N \leq R^2/\rho^2 + 1$. When N is odd, a similar but more careful analysis (see Problem 8.8) shows that $N \leq R^2/\rho^2 + 1 + \frac{1}{N}$. In both cases, $N \leq \lceil R^2/\rho^2 \rceil + 1$.

Assume N is even. We need the following geometric fact (which is proved in Problem 8.7). There exists a (balanced) dichotomy $y_1, \ldots, y_n$ such that

$$\sum_{n=1}^{N} y_n = 0, \text{ and } \qquad \left\| \sum_{n=1}^{N} y_n \mathbf{x}_n \right\| \leq \frac{NR}{\sqrt{N-1}}. \tag{8.7}$$

(For random y_n, $\sum_n y_n \mathbf{x}_n$ is a random walk whose distance from the origin doesn't grow faster than $R\sqrt{N}$.) The dichotomy satisfying (8.7) is separated with margin at least ρ (since $\mathbf{x}_1, \ldots, \mathbf{x}_N$ is shattered). So, for some $(\mathbf{w}, b)$,

$$\rho\|\mathbf{w}\| \leq y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b), \qquad \text{for } n = 1, \ldots, N.$$

Summing over n, using $\sum_n y_n = 0$ and the Cauchy-Schwarz inequality,

$$N\rho\|\mathbf{w}\| \leq \mathbf{w}^{\mathrm{T}} \sum_{n=1}^{N} y_n \mathbf{x}_n + b \sum_{n=1}^{N} y_n = \mathbf{w}^{\mathrm{T}} \sum_{n=1}^{N} y_n \mathbf{x}_n \leq \|\mathbf{w}\| \left\| \sum_{n=1}^{N} y_n \mathbf{x}_n \right\|.$$

By the bound in (8.7), the RHS is at most $\|\mathbf{w}\|NR/\sqrt{N-1}$, or:

$$\rho \leq \frac{R}{\sqrt{N-1}} \implies N \leq \frac{R^2}{\rho^2} + 1.$$

■

End safe skip: Welcome back for a summary.

Combining Theorem 8.5 with $d_{\text{VC}}(\rho) \leq d_{\text{VC}}(0) = d + 1$, we have that

$$d_{\text{VC}}(\rho) \leq \min\left(\lceil R^2/\rho^2 \rceil, d\right) + 1.$$

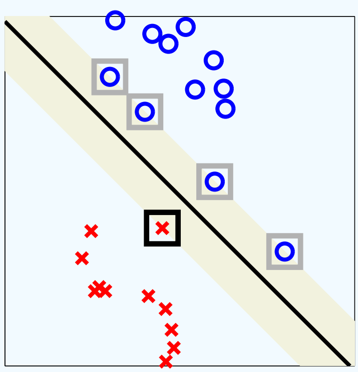
The bound suggests that ρ can be used to control model complexity. The separating hyperplane ($E_{\text{in}} = 0$) with the maximum margin ρ will have the smallest $d_{\text{VC}}(\rho)$ and hence smallest generalization error bar.

Unfortunately, there is a complication. The correct width ρ to use is not known to us ahead of time (what if we chose a higher ρ than possible?). The optimal hyperplane algorithm fixes ρ only after seeing the data. Giving yourself the option to use a smaller ρ but settling on a higher ρ when you see the data means the data is being snooped. One needs to modify the VC analysis to take into account this kind of data snooping. The interested reader can find the details related to these technicalities in the literature.

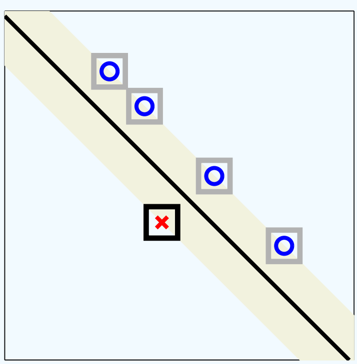
(iii) Bounding the Cross Validation Error. In Chapter 4, we introduced the leave-one-out cross validation error E_{cv} which is the average of the leave-one-out errors e_n. Each e_n is computed by leaving out the data point $(\mathbf{x}_n, y_n)$ and learning on the remaining data to obtain g_n^-. The hypothesis g_n^- is evaluated on $(\mathbf{x}_n, y_n)$ to give e_n, and

$$E_{\text{cv}} = \frac{1}{N}\sum_{n=1}^{N} \mathsf{e}_n.$$

An important property of E_{cv} is that it is an unbiased estimate of the expected out-of-sample error for a data set of size $N-1$. (see Chapter 4 for details on validation). Hence, E_{cv} is a very useful proxy for E_{out}. We can get a surprisingly simple bound for E_{cv} using the number of support vectors. Recall that a support vector lies on the very edge of the margin of the optimal hyperplane. We illustrate a toy data set in Figure 8.5(a) with the support vectors highlighted in boxes.



(a) All data



(b) Only support vectors

Figure 8.5: (a) Optimal hyperplane with support vectors enclosed in boxes. (b) The same hyperplane is obtained using only the support vectors.

The crucial observation as illustrated in Figure 8.5(b) is that if any (or all) the data points other than the support vectors are removed, the resulting separator produced by the SVM is unchanged. You are asked to give a formal proof of this in Problem 8.9, but Figure 8.5 should serve as ample justification. This observation has an important implication: $\mathsf{e}_n = 0$ for any data point $(\mathbf{x}_n, y_n)$ that is not a support vector. This is because removing that data point results in the same separator, and since $(\mathbf{x}_n, y_n)$ was classified correctly before removal, it will remain correct after removal. For the support vectors, $\mathsf{e}_n \le 1$ (binary

classification error), and so

$$E_{\text{cv}}(\text{SVM}) = \frac{1}{N}\sum_{n=1}^{N} \mathsf{e}_n \leq \frac{\#\text{ support vectors}}{N}. \tag{8.8}$$

Exercise 8.8

(a) Evaluate the bound in (8.8) for the data in Figure 8.5.

(b) If one of the four support vectors in a gray box are removed, does the classifier change?

(c) Use your answer in (b) to improve your bound in (a).

The support vectors in gray boxes are non-essential and the support vector in the black box is essential. One can improve the bound in (8.8) to use only essential support vectors. The number of support vectors is unbounded, but the number of essential support vectors is at most $d+1$ (usually much less).

In the interest of full disclosure, and out of fairness to the PLA, we should note that a bound on E_{cv} can be obtained for PLA as well, namely

$$E_{\text{cv}}(\text{PLA}) \leq \frac{R^2}{N\rho^2},$$

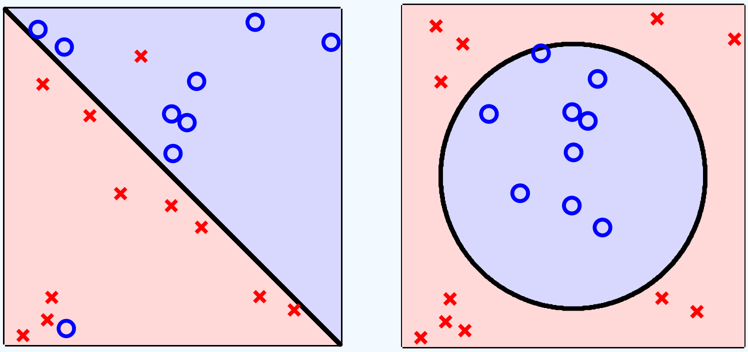
where ρ is the margin of the thickest hyperplane that separates the data, and R is an upper bound on $\|\mathbf{x}_n\|$ (see Problem 8.11). The table below provides a summary of what we know based on our discussion so far.

Algorithm For Selecting Separating Hyperplane		
General	PLA	SVM (Optimal Hyperplane)
$d_{\text{VC}} = d+1$		$d_{\text{VC}}(\rho) \leq \min\left(\left\lceil \frac{R^2}{\rho^2} \right\rceil, d\right) + 1$
	$E_{\text{cv}} \leq \frac{R^2}{N\rho^2}$	$E_{\text{cv}} \leq \frac{\#\text{ support vectors}}{N}$

In general, all you can conclude is the VC bound based on a VC dimension of $d+1$. In high dimensions, this bound can be a very loose. For PLA or SVM, we have additional bounds that do not explicitly depend on the dimension d. If the margin is large, or if the number of support vectors is small (even in infinite dimensions), we are still in good shape.

8.1.3 Non-Separable Data

Our entire discussion so far assumed that the data is linearly separable and focused on separating the data with maximum safety cushion. What if the



(a) Few noisy data. (b) Nonlinearly separable.

Figure 8.6: Non-separable data (reproduction of Figure 3.1).

data is not linearly separable? Figure 8.6 (reproduced from Chapter 3) illustrates the two types of non-separability. In Figure 8.6(a), two noisy data points render the data non-separable. In Figure 8.6(b), the target function is inherently nonlinear.

For the learning problem in Figure 8.6(a), we prefer the linear separator, and need to tolerate the few noisy data points. In Chapter 3, we modified the PLA into the pocket algorithm to handle this situation. Similarly, for SVMs, we will modify the hard-margin SVM to the *soft-margin* SVM in Section 8.4. Unlike the hard margin SVM, the soft-margin SVM allows data points to violate the cushion, or even be misclassified.

To address the other situation in Figure 8.6(b), we introduced the nonlinear transform in Chapter 3. There is nothing to stop us from using the nonlinear transform with the optimal hyperplane, which we will do here.

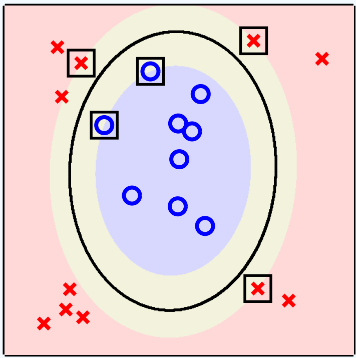
To render the data separable, we would typically transform into a higher dimension. Consider a transform $\Phi\colon \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$. The transformed data are

$$\mathbf{z}_n = \mathbf{\Phi}(\mathbf{x}_n).$$

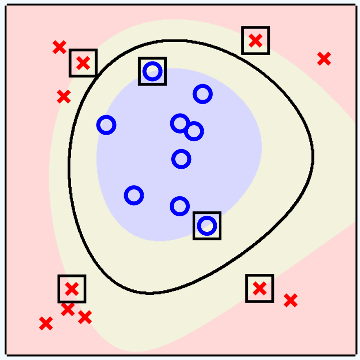
After transforming the data, we solve the hard-margin SVM problem in the $\mathcal{Z}$ space, which is just (8.4) written with $\mathbf{z}_n$ instead of $\mathbf{x}_n$:

$$\begin{aligned} \underset{\tilde{b},\tilde{\mathbf{w}}}{\text{minimize:}} \quad & \frac{1}{2}\tilde{\mathbf{w}}^{\mathrm{T}}\tilde{\mathbf{w}} \\ \text{subject to:} \quad & y_n\left(\tilde{\mathbf{w}}^{\mathrm{T}}\mathbf{z}_n + \tilde{b}\right) \geq 1 \qquad (n = 1, \cdots, N), \end{aligned} \tag{8.9}$$

where $\tilde{\mathbf{w}}$ is now in $\mathbb{R}^{\tilde{d}}$ instead of $\mathbb{R}^d$ (recall that we use tilde for objects in $\mathcal{Z}$ space). The optimization problem in (8.9) is a QP-problem with $\tilde{d} + 1$



(a) SVM with Φ_2



(b) SVM with Φ_3

Figure 8.7: Nonlinear separation using the SVM with 2nd and 3rd order polynomial transforms. The margin is shaded in yellow, and the support vectors are boxed. The dimension of Φ_3 is nearly double that of Φ_2, yet the resulting SVM separator is not severely overfitting with Φ_3.

optimization variables and N constraints. To solve this optimization problem, we can use the standard hard-margin algorithm in the algorithm box on page 8-10, after we replace $\mathbf{x}_n$ with $\mathbf{z}_n$ and d with $\tilde{d}$. The algorithm returns an optimal solution $\tilde{b}^*, \tilde{\mathbf{w}}^*$ and the final hypothesis is

$$g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^{*\text{T}}\Phi(\mathbf{x}) + \tilde{b}^*).$$

In Chapter 3, we introduced a general kth-order polynomial transform Φ_k. For example, the second order polynomial transform is

$$\Phi_2(\mathbf{x}) = (x_1, x_2, x_1^2, x_1x_2, x_2^2).$$

Figure 8.7 shows the result of using the SVM with the 2nd and 3rd order polynomial transforms Φ_2, Φ_3 for the data in Figure 8.6(b). Observe that the 'margin' does not have constant width in the $\mathcal{X}$ space; it is in the $\mathcal{Z}$ space that the width of the separator is uniform. Also, in Figure 8.7(b) you can see that some blue points near the top left are not support vectors, even though they appear closer to the separating surface than the red support vectors near the bottom right. Again this is because it is distances to the separator in the $\mathcal{Z}$ space that matter. Lastly, the dimensions of the two feature spaces are $\tilde{d}_2 = 5$ and $\tilde{d}_3 = 9$ (almost double for the 3rd order polynomial transform). However, the number of support vectors increased from 5 to only 6, and so the bound on E_{cv} did not nearly double.

The benefit of the nonlinear transform is that we can use a sophisticated boundary to lower E_{in}. However, you pay a price to get this sophisticated boundary in terms of a larger d_{VC} and a tendency to overfit. This trade-off is highlighted in the table below for PLA, to compare with SVM.

	perceptrons	perceptron + nonlinear transform
complexity control	**small d_{vc}**	large d_{vc}
boundary	linear	**sophisticated**

We observe from Figure 8.7(b) that the 3rd order polynomial SVM does not show the level of overfitting we might expect when we nearly double the number of free parameters. We can have our cake and eat it too: we enjoy the benefit of high-dimensional transforms in terms of getting sophisticated boundaries, and yet we don't pay too much in terms of E_{out} because d_{vc} and E_{cv} can be controlled in terms of quantities not directly linked to $\tilde{d}$. The table illustrating the trade-offs for the SVM is:

	SVM	SVM + nonlinear transform
complexity control	**smaller 'effective' d_{vc}**	**not too large 'effective' d_{vc}**
boundary	linear	**sophisticated**

You now have the support vector machine at your fingertips: a very powerful, easy to use linear model which comes with automatic regularization. We could be done, but instead, we are going to introduce you to a very powerful tool that can be used to implement nonlinear transforms called a *kernel*. The kernel will allow you to fully exploit the capabilities of the SVM. The SVM has a potential robustness to overfitting even after transforming to a much higher dimension that opens up a new world of possibilities: what about using infinite dimensional transforms? Yes, infinite! It is certainly not feasible within our current technology to use an infinite dimensional transform; but, by using the 'kernel trick', not only can we make infinite dimensional transforms feasible, we can also make them efficient. Stay tuned; it's exciting stuff.

8.2 Dual Formulation of the SVM

The promise of infinite dimensional nonlinear transforms certainly whets the appetite, but we are going to have to earn our cookie ☹. We are going to introduce the *dual* SVM problem which is equivalent to the original *primal* problem (8.9) in that solving both problems gives the same optimal hyperplane, but the dual problem is often easier. It is this dual (no pun intended) view of the SVM that will enable us to exploit the kernel trick that we have touted so loudly. But, the dual view also stands alone as an important formulation of the SVM that will give us further insights into the optimal hyperplane. The connection between primal and dual optimization problems is a heavily studied subject in optimization theory, and we only introduce those few pieces that we need for the SVM.

Note that (8.9) is a QP-problem with $\tilde{d}+1$ variables $(\tilde{b}, \tilde{\mathbf{w}})$ and N constraints. It is computationally difficult to solve (8.9) when $\tilde{d}$ is large, let alone

infinite. The dual problem will also be a QP-problem, but with N variables and $N+1$ constraints – the computational burden no longer depends on $\tilde{d}$, which is a big saving when we move to very high $\tilde{d}$.

Deriving the dual is not going to be easy, but on the positive side, we have already seen the main tool that we will need (in Section 4.2 when we talked about regularization). Regularization introduced us to the constrained minimization problem

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize:}} \quad & E_{\text{in}}(\mathbf{w}) \\ \text{subject to:} \quad & \mathbf{w}^{\mathrm{T}}\mathbf{w} \leq C, \end{aligned}$$

where C is a user-specified parameter. We showed that for a given C, there is a λ such that minimizing the augmented error $E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \lambda\mathbf{w}^{\mathrm{T}}\mathbf{w}$ gives the same regularized solution. We viewed the *Lagrange multiplier* λ as the user-specified parameter instead of C, and minimized the augmented error instead of solving the constrained minimization problem. Here, we are also going to use Lagrange multipliers, but in a slightly different way. The Lagrange multipliers will arise as *variables* that correspond to the constraints, and we need to formulate a new optimization problem (which is the dual problem) to solve for those variables.

8.2.1 Lagrange Dual for a QP-Problem

Let us illustrate the concept of the dual with a simplified version of the standard QP-problem, using just one inequality constraint. All the concepts will easily generalize to the case with more than one constraint. So, consider the QP-problem

$$\begin{aligned} \underset{\mathbf{u}\in\mathbb{R}^L}{\text{minimize:}} \quad & \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} \\ \text{subject to:} \quad & \mathbf{a}^{\mathrm{T}}\mathbf{u} \geq c \end{aligned} \tag{8.10}$$

Here is a closely related optimization problem.

$$\underset{\mathbf{u}\in\mathbb{R}^L}{\text{minimize:}} \quad \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} + \max_{\alpha\geq 0} \alpha\left(c - \mathbf{a}^{\mathrm{T}}\mathbf{u}\right). \tag{8.11}$$

The variable $\alpha \geq 0$ multiplies the constraint $(c - \mathbf{a}^{\mathrm{T}}\mathbf{u})$.[4] To obtain the objective in (8.11) from (8.10), we add what amounts to a penalty term that encourages $(c - \mathbf{a}^{\mathrm{T}}\mathbf{u})$ to be negative and satisfy the constraint. The optimization problem in (8.10) is equivalent to the one in (8.11) as long as there is at least one solution that satisfies the constraint in (8.10). The advantage in (8.11) is that the minimization with respect to $\mathbf{u}$ is *unconstrained*, and the price is a slightly more complex objective that involves this 'Lagrangian penalty term'. The next exercise proves the equivalence.

[4] The parameter α is called a Lagrange multiplier. In the optimization literature, the Lagrange multiplier would typically be denoted by λ. Historically, the SVM literature has used α, which is the convention we follow.

Exercise 8.9

Let $\mathbf{u}_0$ be optimal for (8.10), and let $\mathbf{u}_1$ be optimal for (8.11).

(a) Show that $\max_{\alpha\geq 0} \alpha\,(c - \mathbf{a}^{\mathrm{T}}\mathbf{u}_0) = 0$. *[Hint: $c - \mathbf{a}^{\mathrm{T}}\mathbf{u}_0 \leq 0$.]*

(b) Show that $\mathbf{u}_1$ is feasible for (8.10). To show this, suppose to the contrary that $c - \mathbf{a}^{\mathrm{T}}\mathbf{u}_1 > 0$. Show that the objective in (8.11) is infinite, whereas $\mathbf{u}_0$ attains a finite objective of $\frac{1}{2}\mathbf{u}_0^{\mathrm{T}}\mathrm{Q}\mathbf{u}_0 + \mathbf{p}^{\mathrm{T}}\mathbf{u}_0$, which contradicts the optimality of $\mathbf{u}_1$.

(c) Show that $\frac{1}{2}\mathbf{u}_1^{\mathrm{T}}\mathrm{Q}\mathbf{u}_1 + \mathbf{p}^{\mathrm{T}}\mathbf{u}_1 = \frac{1}{2}\mathbf{u}_0^{\mathrm{T}}\mathrm{Q}\mathbf{u}_0 + \mathbf{p}^{\mathrm{T}}\mathbf{u}_0$, and hence that $\mathbf{u}_1$ is optimal for (8.10) and $\mathbf{u}_0$ is optimal for (8.11).

(d) Let $\mathbf{u}^*$ be any optimal solution for (8.11) with $\max_{\alpha\geq 0} \alpha(c - \mathbf{a}^{\mathrm{T}}\mathbf{u}^*)$ attained at α^*. Show that

$$\alpha^*(c - \mathbf{a}^{\mathrm{T}}\mathbf{u}^*) = 0. \tag{8.12}$$

Either the constraint is exactly satisfied with $c - \mathbf{a}^{\mathrm{T}}\mathbf{u}^* = 0$, or $\alpha^* = 0$.

Exercise 8.9 shows that as long as the original problem in (8.10) is feasible, we can obtain a solution by solving (8.11) instead. Let us analyze (8.11) in further detail. Our goal is to simplify it. Introduce the Lagrangian function $\mathcal{L}(\mathbf{u}, \alpha)$, defined by

$$\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha}) = \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} + \alpha\,(c - \mathbf{a}^{\mathrm{T}}\mathbf{u}). \tag{8.13}$$

In terms of $\mathcal{L}$, the optimization problem in (8.11) is

$$\min_{\mathbf{u}} \max_{\alpha\geq 0} \mathcal{L}(\mathbf{u}, \alpha). \tag{8.14}$$

For convex quadratic programming, when $\mathcal{L}(\mathbf{u}, \alpha)$ has the special form in (8.13) and $c - \mathbf{a}^{\mathrm{T}}\mathbf{u} \leq 0$ is feasible, a profound relationship known as *strong duality* has been proven to hold:

$$\min_{\mathbf{u}} \max_{\alpha\geq 0} \mathcal{L}(\mathbf{u}, \alpha) = \max_{\alpha\geq 0} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \alpha). \tag{8.15}$$

An interested reader can consult a standard reference in convex optimization for a proof. The impact for us is that a solution to the optimization problem on the RHS of (8.15) gives a solution to the problem on the LHS, which is the problem we want to solve. This helps us because on the RHS, one is first minimizing with respect to an *unconstrained* $\mathbf{u}$, and that we can do analytically. This analytical step considerably reduces the complexity of the problem. Solving the problem on the RHS of (8.15) is known as solving the *Lagrange dual problem* (dual problem for short). The original problem is called the *primal* problem.

We briefly discuss what to do when there are many constraints, $\mathbf{a}_m^{\mathrm{T}}\mathbf{u} \geq c_m$ for $m = 1, \ldots, M$. Not much changes. All we do is introduce a Lagrange

multiplier $\alpha_m \geq 0$ for *each* constraint and add the penalty term $\alpha_m(c_m - \mathbf{a}_m^{\mathrm{T}}\mathbf{u})$ into the Lagrangian. Then, just as before, we solve the dual problem. A simple example will illustrate all the mechanics.

Example 8.6. Let's minimize $u_1^2 + u_2^2$ subject to $u_1 + 2u_2 \geq 2$ and $u_1, u_2 \geq 0$. We first construct the Lagrangian,

$$\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha}) = u_1^2 + u_2^2 + \alpha_1(2 - u_1 - 2u_2) - \alpha_2 u_1 - \alpha_3 u_2,$$

where, as you can see, we have added a penalty term for each of the three constraints, and each penalty term has an associated Lagrange multiplier. To solve the dual optimization problem, we first need to minimize $\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})$ with respect to the unconstrained $\mathbf{u}$. We then need to maximize with respect to $\boldsymbol{\alpha} \geq \mathbf{0}$. To do the unconstrained minimization with respect to $\mathbf{u}$, we use the standard first derivative condition from calculus:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial u_1} = 0 \quad &\Longrightarrow \quad u_1 = \frac{\alpha_1 + \alpha_2}{2}; \\ \frac{\partial \mathcal{L}}{\partial u_2} = 0 \quad &\Longrightarrow \quad u_2 = \frac{2\alpha_1 + \alpha_3}{2}. \end{aligned} \tag{$*$}$$

Plugging these values for u_1 and u_2 back into $\mathcal{L}$ and collecting terms, we have a function only of $\boldsymbol{\alpha}$, which we have to maximize with respect to $\alpha_1, \alpha_2, \alpha_3$:

$$\begin{aligned} \text{maximize:} \quad & \mathcal{L}(\boldsymbol{\alpha}) = -\tfrac{5}{4}\alpha_1^2 - \tfrac{1}{4}\alpha_2^2 - \tfrac{1}{4}\alpha_3^2 - \tfrac{1}{2}\alpha_1\alpha_2 - \alpha_1\alpha_3 + 2\alpha_1 \\ \text{subject to:} \quad & \alpha_1, \alpha_2, \alpha_3 \geq 0 \end{aligned}$$

Exercise 8.10

Do the algebra. Derive $(*)$ and plug it into $\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})$ to obtain $\mathcal{L}(\boldsymbol{\alpha})$.

By going to the dual, all we accomplished is to obtain another QP-problem! So, in a sense, we have not solved the problem at all. What did we gain? The new problem is easier to solve. This is because the constraints for the dual problem are simple ($\boldsymbol{\alpha} \geq \mathbf{0}$). In our case, all terms involving α_2 and α_3 are at most zero, and we can attain zero by setting $\alpha_2 = \alpha_3 = 0$. This leaves us with $-\frac{5}{4}\alpha_1^2 + 2\alpha_1$, which is maximized when $\alpha_1 = \frac{4}{5}$. So the final solution is $\alpha_1 = \frac{4}{5}$, $\alpha_2 = \alpha_3 = 0$, and plugging these into $(*)$ gives

$$u_1 = \tfrac{2}{5} \qquad \text{and} \qquad u_2 = \tfrac{4}{5}.$$

The optimal value for the objective is $u_1^2 + u_2^2 = \frac{4}{5}$. □

We summarize our discussion in the following theorem, which states the dual formulation of a QP-problem. In the next section, we will show how this dual formulation is applied to the SVM QP-problem. The theorem looks formidable, but don't be intimidated. Its application to the SVM QP-problem is conceptually no different from our little example above.

Theorem 8.7 (KKT)**.** For a feasible convex QP-problem in *primal* form,

$$\begin{aligned} \underset{\mathbf{u}\in\mathbb{R}^L}{\text{minimize:}} \quad & \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} \\ \text{subject to:} \quad & \mathbf{a}_m^{\mathrm{T}}\mathbf{u} \geq c_m \qquad (m = 1, \cdots, M), \end{aligned}$$

define the Lagrange function

$$\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha}) = \tfrac{1}{2}\mathbf{u}^{\mathrm{T}}\mathrm{Q}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{u} + \sum_{m=1}^{M} \alpha_m \left(c_m - \mathbf{a}_m^{\mathrm{T}}\mathbf{u}\right).$$

The solution $\mathbf{u}^*$ is optimal for the primal if and only if $(\mathbf{u}^*, \boldsymbol{\alpha}^*)$ is a solution to the dual optimization problem

$$\max_{\boldsymbol{\alpha}\geq\mathbf{0}} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\alpha}).$$

The optimal $(\mathbf{u}^*, \boldsymbol{\alpha}^*)$ satisfies the Karush-Kühn-Tucker (KKT) conditions:

(i) *Primal and dual constraints:*

$$\mathbf{a}_m^{\mathrm{T}}\mathbf{u}^* \geq c_m \quad \text{and} \quad \alpha_m \geq 0 \qquad (m = 1, \cdots, M).$$

(ii) *Complementary slackness:*

$$\alpha_m^* \left(\mathbf{a}_m^{\mathrm{T}}\mathbf{u}^* - c_m\right) = 0.$$

(iii) *Stationarity with respect to* $\mathbf{u}$*:*

$$\nabla_{\mathbf{u}}\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})\big|_{\mathbf{u}=\mathbf{u}^*, \boldsymbol{\alpha}=\boldsymbol{\alpha}^*} = \mathbf{0}.$$

The three KKT conditions in Theorem 8.7 characterize the relationship between the optimal $\mathbf{u}^*$ and $\boldsymbol{\alpha}^*$ and can often be used to simplify the Lagrange dual problem. The constraints are inherited from the constraints in the definitions of the primal and dual optimization problems. Complementary slackness is a condition you derived in (8.12) which says that at the optimal solution, the constraints fall into two types: those which are on the boundary and are exactly satisfied (the *active* constraints) and those which are in the interior of the feasible set. The interior constraints must have Lagrange multipliers equal to zero. The stationarity with respect to $\mathbf{u}$ is the necessary and sufficient condition for a convex program to solve the first part of the dual problem, namely $\min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})$. Next, we focus on the hard-margin SVM, and use the KKT conditions, in particular, stationarity with respect to $\mathbf{u}$, to simplify the Lagrange dual problem.

8.2.2 Dual of the Hard-Margin SVM

We now apply the KKT theorem to the convex QP-problem for hard-margin SVM (8.9). The mechanics of our derivation are not more complex than in

Example 8.6, though the algebra is more cumbersome. The steps we followed in Example 8.6 are a helpful guide to keep in mind.

The hard-margin optimization problem only applies when the data is linearly separable. This means that the optimization problem is convex and feasible, so the KKT theorem applies. For your convenience, we reproduce the hard-margin SVM QP-problem here,

$$\begin{aligned} \underset{b,\mathbf{w}}{\text{minimize:}} \quad & \tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \\ \text{subject to:} \quad & y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 \qquad (n = 1, \cdots, N). \end{aligned} \tag{8.16}$$

The optimization variable is $\mathbf{u} = \left[\begin{smallmatrix} b \\ \mathbf{w} \end{smallmatrix}\right]$. The first task is to construct the Lagrangian. There are N constraints, so we add N penalty terms, and introduce a Lagrange multiplier α_n for each penalty term. The Lagrangian is

$$\begin{aligned} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) &= \tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \sum_{n=1}^{N} \alpha_n \left(1 - y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)\right) \\ &= \tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n - b\sum_{n=1}^{N} \alpha_n y_n + \sum_{n=1}^{N} \alpha_n. \end{aligned} \tag{8.17}$$

We must first minimize $\mathcal{L}$ with respect to $(b, \mathbf{w})$ and then maximize with respect to $\boldsymbol{\alpha} \geq \mathbf{0}$. To minimize with respect to $(b, \mathbf{w})$, we need the derivatives of $\mathcal{L}$. The derivative with respect to b is just the coefficient of b because b appears linearly in the Lagrangian. To differentiate the terms involving $\mathbf{w}$, we need some vector calculus identities from the linear algebra appendix: $\frac{\partial}{\partial \mathbf{w}}\mathbf{w}^{\mathrm{T}}\mathbf{w} = 2\mathbf{w}$ and $\frac{\partial}{\partial \mathbf{w}}\mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{x}$. The reader may now verify that

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n \qquad \text{and} \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n.$$

Setting these derivatives to zero gives

$$\sum_{n=1}^{N} \alpha_n y_n = 0; \tag{8.18}$$

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n. \tag{8.19}$$

Something strange has happened (highlighted in red), which did not happen in Example 8.6. After setting the derivatives to zero in Example 8.6, we were able to solve for $\mathbf{u}$ in terms of the Lagrange multipliers $\boldsymbol{\alpha}$. Here, the stationarity condition for b does not allow us to solve for b directly ☹. Instead, we got a *constraint* that $\boldsymbol{\alpha}$ must satisfy at the final solution. Don't let this unsettle you. The KKT theorem will allow us to ultimately recover b ☺. This constraint on $\boldsymbol{\alpha}$ is not surprising. Any choice for $\boldsymbol{\alpha}$ which does not satisfy (8.18) would

allow $\mathcal{L} \to -\infty$ by appropriately choosing b. Since we *maximize* over $\boldsymbol{\alpha}$, we must choose $\boldsymbol{\alpha}$ to satisfy the constraint and prevent $\mathcal{L} \to -\infty$.

We proceed as in Example 8.6 by plugging the stationarity constraints back into the Lagrangian to get a function solely in terms of $\boldsymbol{\alpha}$. Observe that the constraint in (8.18) means that the term involving b in the Lagrangian (8.17) reduces to zero. The terms in the Lagrangian (8.17) involving the weights $\mathbf{w}$ simplify when we use the expression in (8.19):

$$
\begin{aligned}
&\tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n \\
&= \frac{1}{2} \underbrace{\sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^{\mathrm{T}}}_{\mathbf{w}^{\mathrm{T}}} \underbrace{\sum_{m=1}^{N} \alpha_m y_m \mathbf{x}_m}_{\mathbf{w}} - \sum_{n=1}^{N} \alpha_n y_n \underbrace{\sum_{m=1}^{N} \alpha_m y_m \mathbf{x}_m^{\mathrm{T}}}_{\mathbf{w}^{\mathrm{T}}} \mathbf{x}_n \\
&= \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m - \sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m \\
&= -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m. \qquad (8.20)
\end{aligned}
$$

After we use (8.18) and (8.20) in (8.17), the Lagrangian reduces to a simpler function of just the variable $\boldsymbol{\alpha}$:

$$
\mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m + \sum_{n=1}^{N} \alpha_n.
$$

We must now maximize $\mathcal{L}(\boldsymbol{\alpha})$ subject to $\boldsymbol{\alpha} \geq \mathbf{0}$, and we cannot forget the constraint (8.18) which we inherited from the stationarity condition on b. We can equivalently *minimize* the negative of $\mathcal{L}(\boldsymbol{\alpha})$, and so we have the following optimization problem to solve.

$$
\begin{aligned}
\underset{\boldsymbol{\alpha}\in\mathbb{R}^N}{\text{minimize:}} \quad & \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m - \sum_{n=1}^{N} \alpha_n \qquad (8.21) \\
\text{subject to:} \quad & \sum_{n=1}^{N} y_n \alpha_n = 0 \\
& \alpha_n \geq 0 \qquad (n = 1, \cdots, N).
\end{aligned}
$$

If you made it to here, you deserve a pat on the back for surviving all the algebra ☺. The result is similar to what happened in Example 8.6. We have not solved the problem; we have reduced it to another QP-problem. The next exercise guides you through the steps to put (8.21) into the standard QP form.

Exercise 8.11

(a) Show that the problem in (8.21) is a standard QP-problem:

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^N}{\text{minimize:}} \quad & \tfrac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathrm{Q}_{\mathrm{D}}\boldsymbol{\alpha} - \mathbf{1}_N^{\mathrm{T}}\boldsymbol{\alpha} \\ \text{subject to:} \quad & \mathrm{A}_{\mathrm{D}}\boldsymbol{\alpha} \geq \mathbf{0}_{N+2}, \end{aligned} \tag{8.22}$$

where Q_{D} and A_{D} (D for dual) are given by:

$$\mathrm{Q}_{\mathrm{D}} = \begin{bmatrix} y_1y_1\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 & \dots & y_1y_N\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_N \\ y_2y_1\mathbf{x}_2^{\mathrm{T}}\mathbf{x}_1 & \dots & y_2y_N\mathbf{x}_2^{\mathrm{T}}\mathbf{x}_N \\ \vdots & \vdots & \vdots \\ y_Ny_1\mathbf{x}_N^{\mathrm{T}}\mathbf{x}_1 & \dots & y_Ny_N\mathbf{x}_N^{\mathrm{T}}\mathbf{x}_N \end{bmatrix} \text{ and } \mathrm{A}_{\mathrm{D}} = \begin{bmatrix} \mathbf{y}^{\mathrm{T}} \\ -\mathbf{y}^{\mathrm{T}} \\ \mathrm{I}_{N\times N} \end{bmatrix}.$$

[Hint: Recall that an equality corresponds to two inequalities.]

(b) The matrix Q_{D} of quadratic coefficients is $[\mathrm{Q}_{\mathrm{D}}]_{mn} = y_my_n\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n$. Show that $\mathrm{Q}_{\mathrm{D}} = \mathrm{X}_s\mathrm{X}_s^{\mathrm{T}}$, where X_s is the 'signed data matrix',

$$\mathrm{X}_s = \begin{bmatrix} — & y_1\mathbf{x}_1^{\mathrm{T}} & — \\ — & y_2\mathbf{x}_2^{\mathrm{T}} & — \\ & \vdots & \\ — & y_N\mathbf{x}_N^{\mathrm{T}} & — \end{bmatrix}.$$

Hence, show that Q_{D} is positive semi-definite. This implies that the QP-problem is convex.

Example 8.8. Let us illustrate all the machinery of the dual formulation using the data in Example 8.2 (the toy problem from Section 8.1). For your convenience, we reproduce the data, and we compute $\mathrm{Q}_{\mathrm{D}}, \mathrm{A}_{\mathrm{D}}$ using Exercise 8.11:

$$\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 2 & 0 \\ 3 & 0 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix}; \quad \mathrm{Q}_{\mathrm{D}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 8 & -4 & -6 \\ 0 & -4 & 4 & 6 \\ 0 & -6 & 6 & 9 \end{bmatrix}, \quad \mathrm{A}_{\mathrm{D}} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We can now write down the optimization problem in (8.21) and manually solve it to get the optimal $\boldsymbol{\alpha}$. The masochist reader is invited to do so in Problem 8.3. Instead, we can use a QP-solver to solve (8.22) in under a millisecond:

$$\boldsymbol{\alpha}^* \leftarrow \mathsf{QP}(\mathrm{Q}_{\mathrm{D}}, -\mathbf{1}, \mathrm{A}_{\mathrm{D}}, \mathbf{0}) \qquad \text{gives} \qquad \boldsymbol{\alpha}^* = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 1 \\ 0 \end{bmatrix}.$$

Now that we have $\boldsymbol{\alpha}^*$, we can compute the optimal weights using (8.19),

$$\mathbf{w}^* = \sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n = -\tfrac{1}{2}\begin{bmatrix}0\\0\end{bmatrix} - \tfrac{1}{2}\begin{bmatrix}2\\2\end{bmatrix} + \begin{bmatrix}2\\0\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}.$$

As expected, these are the same optimal weights we got in Example 8.2. What about b? This is where things get tricky. Recall the complementary slackness KKT condition in Theorem 8.7. It asserts that if $\alpha_n^* > 0$, then the corresponding constraint must be exactly satisfied. In our example, $\alpha_1 = \frac{1}{2} > 0$, so $y_1(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_1 + b^*) = 1$. Since $\mathbf{x}_1 = \mathbf{0}$ and $y_1 = -1$, this means $b^* = -1$, exactly as in Example 8.2. So, $g(\mathbf{x}) = \text{sign}(x_1 - x_2 - 1)$. □

On a practical note, the $N \times N$ matrix Q_D is often dense (containing lots of non-zero elements). If $N = 100,000$, storing Q_D uses more than 10GB of RAM. Thus, for large-scale applications, specially tailored QP packages that dynamically compute Q_D and use specific properties of SVM are often used to solve (8.22) efficiently.

8.2.3 Recovering the SVM from the Dual Solution

Now that we have solved the dual problem using quadratic programming to obtain the optimal solution $\boldsymbol{\alpha}^*$, what remains is to compute the optimal hyperplane $(b^*, \mathbf{w}^*)$. The weights are easy, and are obtained using the stationary condition in (8.19):

$$\mathbf{w}^* = \sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n. \tag{8.23}$$

Assume that the data contains at least one positive and one negative example (otherwise the classification problem is trivial). Then, at least one of the $\{\alpha_n^*\}$ will be strictly positive. Let $\alpha_s^* > 0$ (s for *support* vector). The KKT complementary slackness condition in Theorem 8.7 tells us that the constraint corresponding to this non-zero α_s^* is equality. This means that

$$y_s\left(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_s + b^*\right) = 1.$$

We can solve for b^* in the above equation to get

$$\begin{aligned} b^* &= y_s - \mathbf{w}^{*\mathrm{T}}\mathbf{x}_s \\ &= y_s - \sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_s. \end{aligned} \tag{8.24}$$

Exercise 8.12

If all the data is from one class, then $\alpha_n^* = 0$ for $n = 1, \ldots, N$.

(a) What is $\mathbf{w}^*$?

(b) What is b^*?

Equations (8.23) and (8.24) connect the optimal $\boldsymbol{\alpha}^*$, which we get from solving the dual problem, to the optimal hyperplane $(b^*, \mathbf{w}^*)$ which solves (8.4). Most importantly, the optimal hypothesis is

$$\begin{aligned} g(\mathbf{x}) &= \text{sign}(\mathbf{w}^{*\mathrm{T}}\mathbf{x} + b^*) \\ &= \text{sign}\left(\sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n^{\mathrm{T}}\mathbf{x} + b^*\right) \\ &= \text{sign}\left(\sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n^{\mathrm{T}}(\mathbf{x} - \mathbf{x}_s) + y_s\right). \end{aligned} \tag{8.25}$$

Recall that $(\mathbf{x}_s, y_s)$ is *any* support vector which is defined by the condition $\alpha_s^* > 0$. There is a summation over n in Equations (8.23), (8.24) and (8.25), but only the terms with $\alpha_n^* > 0$ contribute to the summations. We can therefore get a more efficient representation for $g(\mathbf{x})$ using only the positive α_n^*:

$$g(\mathbf{x}) = \text{sign}\left(\sum_{\alpha_n^*>0} y_n \alpha_n^* \mathbf{x}_n^{\mathrm{T}}\mathbf{x} + b^*\right), \tag{8.26}$$

where b^* is given by (8.24) (the summation (8.24) can also be restricted to only those $\alpha_n^* > 0$). So, $g(\mathbf{x})$ is determined by only those examples $(\mathbf{x}_n, y_n)$ for which $\alpha_n^* > 0$. We summarize our long discussion about the dual in the following algorithm box.

Hard-Margin SVM with Dual QP

1: Construct $\mathrm{Q_D}$ and $\mathrm{A_D}$ as in Exercise 8.11

$$\mathrm{Q_D} = \begin{bmatrix} y_1y_1\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 & \dots & y_1y_N\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_N \\ y_2y_1\mathbf{x}_2^{\mathrm{T}}\mathbf{x}_1 & \dots & y_2y_N\mathbf{x}_2^{\mathrm{T}}\mathbf{x}_N \\ \vdots & \vdots & \vdots \\ y_Ny_1\mathbf{x}_N^{\mathrm{T}}\mathbf{x}_1 & \dots & y_Ny_N\mathbf{x}_N^{\mathrm{T}}\mathbf{x}_N \end{bmatrix} \text{ and } \mathrm{A_D} = \begin{bmatrix} \mathbf{y}^{\mathrm{T}} \\ -\mathbf{y}^{\mathrm{T}} \\ \mathrm{I}_{N\times N} \end{bmatrix}.$$

2: Use a QP-solver to optimize the dual problem:

$$\boldsymbol{\alpha}^* \leftarrow \mathsf{QP}(\mathrm{Q_D}, -\mathbf{1}_N, \mathrm{A_D}, \mathbf{0}_{N+2}).$$

3: Let s be a support vector for which $\alpha_s^* > 0$. Compute b^*,

$$b^* = y_s - \sum_{\alpha_n^*>0} y_n \alpha_n^* \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_s.$$

4: Return the final hypothesis

$$g(\mathbf{x}) = \text{sign}\left(\sum_{\alpha_n^*>0} y_n \alpha_n^* \mathbf{x}_n^{\mathrm{T}}\mathbf{x} + b^*\right).$$

The *support vectors* are the examples $(\mathbf{x}_s, y_s)$ for which $\alpha_s^* > 0$. The support vectors play two important roles. First, they serve to identify the data points on boundary of the optimal fat-hyperplane. This is because of the complementary slackness condition in the KKT theorem:

$$y_s \left(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_s + b^*\right) = 1.$$

This condition identifies the support vectors as being closest to, in fact on, the boundary of the optimal fat-hyperplane. The leads us to an interesting geometric interpretation: the dual SVM identifies the support vectors on the boundary of the optimal fat-hyperplane, and uses *only* those support vectors to construct the final classifier. We already highlighted these support vectors in Section 8.1, where we used the term support vector to highlight the fact that these data points are 'supporting' the cushion, preventing it from expanding further.

Exercise 8.13

KKT complementary slackness gives that if $\alpha_n^* > 0$, then $(\mathbf{x}_n, y_n)$ is on the boundary of the optimal fat-hyperplane and $y_n \left(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*\right) = 1$.

Show that the reverse is not true. Namely, it is possible that $\alpha_n^* = 0$ and yet $(\mathbf{x}_n, y_n)$ is on the boundary satisfying $y_n \left(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*\right) = 1$.

[Hint: Consider a toy data set with two positive examples at $(0, 0)$ *and* $(1, 0)$*, and one negative example at* $(0, 1)$*.]*

The previous exercise says that from the dual, we identify a subset of the points on the boundary of the optimal fat-hyperplane which are called support vectors. All points on the boundary are support vector *candidates*, but only those with $\alpha_n^* > 0$ (and contribute to $\mathbf{w}^*$) are the support vectors.

The second role played by the support vectors is to determine the final hypothesis $g(\mathbf{x})$ through (8.26). The SVM dual problem directly identifies the data points relevant to the final hypothesis. Observe that *only* these support vectors are needed to compute $g(\mathbf{x})$.

Example 8.9. In Example 8.8 we computed $\boldsymbol{\alpha}^* = (\frac{1}{2}, \frac{1}{2}, 1, 0)$ for our toy problem. Since α_1^* is positive, we can choose $(\mathbf{x}_1, y_1) = (\mathbf{0}, -1)$ as our support vector to compute b^* in (8.24). The final hypothesis is

$$\begin{aligned} g(\mathbf{x}) &= \operatorname{sign}\left(-\tfrac{1}{2} \cdot \left[\begin{smallmatrix} 2 & 2 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 2 & 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right] - 1\right) \\ &= \operatorname{sign}\left(x_1 - x_2 - 1\right), \end{aligned}$$

computed now for the third time ☺. □

We used the fact that only the support vectors are needed to compute the final hypothesis to derive the upper bound on E_{cv} given in (8.8); this upper bound depends only on the number of support vectors. Actually, our bound

on E_{cv} is based on the number of support vector candidates. Since not all support vector candidates are support vectors, one usually gets a much better bound on E_{cv} by instead using the number of support vectors. That is,

$$E_{\text{cv}} \leq \frac{\text{number of } \alpha_n^* > 0}{N}. \tag{8.27}$$

To prove this bound, it is necessary to show that if one throws out any data point with $\alpha_n = 0$, the final hypothesis is not changed. The next exercise asks you to to do exactly this.

Exercise 8.14

Suppose that we removed a data point $(\mathbf{x}_n, y_n)$ with $\alpha_n^* = 0$.

(a) Show that the previous optimal solution $\boldsymbol{\alpha}^*$ remains feasible for the new dual problem (8.21) (after removing α_n^*) .

(b) Show that if there is any other feasible solution for the new dual that has a lower objective value than $\boldsymbol{\alpha}^*$, this would contradict the optimality of $\boldsymbol{\alpha}^*$ for the original dual problem.

(c) Hence, show that $\boldsymbol{\alpha}^*$ (minus α_n^*) is optimal for the new dual.

(d) Hence, show that the optimal fat-hyperplane did not change.

(e) Prove the bound on E_{cv} in (8.27).

In practice, there typically aren't many support vectors, so the bound in (8.27) can be quite good. Figure 8.8 shows a data set with 50 random data points and the resulting optimal hyperplane with 3 support vectors (in black boxes). The support vectors are identified from the dual solution ($\alpha_n^* > 0$). Figure 8.8

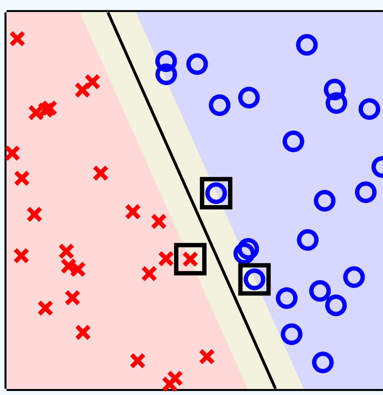


Figure 8.8: Support vectors identified from the SVM dual (3 black boxes).

supports the fact that there are often just a few support vectors. This means that the optimal $\boldsymbol{\alpha}^*$ is usually *sparse*, containing many zero elements and a few non-zeros. The sparsity property means that the representation of $g(\mathbf{x})$ in (8.26) can be computed efficiently using only these few support vectors. If there are many support vectors, it is usually more efficient to compute $(b^*, \mathbf{w}^*)$ ahead of time, and use $g(\mathbf{x}) = \text{sign}(\mathbf{w}^{*\mathrm{T}}\mathbf{x} + b^*)$ for prediction.

8.3 Kernel Trick for SVM

We advertised the kernel as a way to use nonlinear transforms into high dimensional spaces efficiently. We are now going to deliver on that promise for SVM. In order to couple the kernel with SVM, we need to view SVM from the dual formulation. And that is why we expended considerable effort to understand this alternative dual view of SVM. The kernel, together with the dual formulation, will allow us to efficiently run SVM with transforms to high or even infinite dimensional spaces.

8.3.1 Kernel Trick via Dual SVM

Let's start by revisiting the procedure for solving nonlinear SVM from the dual formulation based on a nonlinear transform $\Phi\colon \mathcal{X} \to \mathcal{Z}$, which can be done by replacing $\mathbf{x}$ by $\mathbf{z} = \Phi(\mathbf{x})$ in the algorithm box on page 8-30. First, calculate the coefficients for the dual problem that includes the $\mathrm{Q_D}$ matrix; then solve the dual problem to identify the non-zero Lagrange multipliers α_n^* and the corresponding support vectors $(\mathbf{x}_n, y_n)$; finally, use one of the support vectors to calculate b^*, and return the hypothesis $g(\mathbf{x})$ based on b^*, the support vectors, and their Lagrange multipliers.

Throughout the procedure, the only step that may depend on $\tilde{d}$, which is the dimension of $\Phi(\mathbf{x})$, is in calculating the $\mathcal{Z}$-space inner product

$$\Phi(\mathbf{x})^{\mathrm{T}}\Phi(\mathbf{x}').$$

This inner product is needed in the formulation of $\mathrm{Q_D}$ and in the expression for $g(\mathbf{x})$. The 'kernel trick' is based on the following idea: If the transform *and* the inner product can be jointly and efficiently computed in a way that is independent of $\tilde{d}$, the whole nonlinear SVM procedure can be carried out without computing/storing each $\Phi(\mathbf{x})$ explicitly. Then, the procedure can work efficiently for a large or even an infinite $\tilde{d}$.

So the question is, can we effectively do the transform and compute the inner product in an efficient way irrespective of $\tilde{d}$? Let us first define a function that combines both the transform and the inner product:

$$K_\Phi(\mathbf{x}, \mathbf{x}') \equiv \Phi(\mathbf{x})^{\mathrm{T}}\Phi(\mathbf{x}'). \tag{8.28}$$

This function is called a *kernel function*, or just *kernel*. The kernel takes as input two vectors in the $\mathcal{X}$ space and outputs the inner product that would

be computed in the $\mathcal{Z}$ space (for the transform Φ). In its explicit form (8.28), it appears that the kernel transforms the inputs $\mathbf{x}$ and $\mathbf{x}'$ to the $\mathcal{Z}$ space and then computes the inner product. The efficiency of this process would certainly depend on the dimension of the $\mathcal{Z}$ space, which is $\tilde{d}$. The question is whether we can compute $K_\Phi(\mathbf{x}, \mathbf{x}')$ more efficiently.

For two cases of specific nonlinear transforms Φ, we are going to demonstrate that their corresponding kernel functions K_Φ can indeed be efficiently computed, with a cost proportional to d instead of $\tilde{d}$. (For simplicity, we will use K instead of K_Φ when Φ is clear from the context.)

Polynomial Kernel. Consider the second-order polynomial transform:

$$\Phi_2(\mathbf{x}) \quad = \quad (1, x_1, x_2, \ldots, x_d, x_1x_1, x_1x_2, \ldots, x_dx_d),$$

where $\tilde{d} = 1 + d + d^2$ (the identical features x_ix_j and x_jx_i are included separately for mathematical convenience as you will see below). A direct computation of $\Phi_2(\mathbf{x})$ takes $O(\tilde{d})$ in time, and thus a direct computation of

$$\Phi_2(\mathbf{x})^{\mathrm{T}}\Phi_2(\mathbf{x}') \quad = \quad 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d}\sum_{j=1}^{d} x_i x_j x_i' x_j'$$

also takes time $O(\tilde{d})$. We can simplify the double summation by reorganizing the terms into a product of two separate summations:

$$\sum_{i=1}^{d}\sum_{j=1}^{d} x_i x_j x_i' x_j' = \sum_{i=1}^{d} x_i x_i' \times \sum_{j=1}^{d} x_j x_j' = (\mathbf{x}^{\mathrm{T}}\mathbf{x}')^2.$$

Therefore, we can calculate $\Phi_2(\mathbf{x})^{\mathrm{T}}\Phi_2(\mathbf{x}')$ by an equivalent function

$$K(\mathbf{x}, \mathbf{x}') = 1 + (\mathbf{x}^{\mathrm{T}}\mathbf{x}') + (\mathbf{x}^{\mathrm{T}}\mathbf{x}')^2.$$

In this instance, we see that K can be easily computed in time $O(d)$, which is asymptotically faster than $\tilde{d}$. So, we have demonstrated that, for the polynomial transform, the kernel K is a mathematical and computational shortcut that allows us to combine the transform *and* the inner product into a single more efficient function.

If the kernel K is efficiently computable for some specific Φ, as is the case for our polynomial transform, then whenever we need to compute the inner product of transformed inputs in the $\mathcal{Z}$ space, we can use the *kernel trick* and instead compute the kernel function of those inputs in the $\mathcal{X}$ space. Any learning technique that uses inner products can benefit from this kernel trick.

In particular, let us look back at the SVM dual problem transformed into the $\mathcal{Z}$ space. We obtain the dual problem in the $\mathcal{Z}$ space by replacing every instance of $\mathbf{x}_n$ in (8.21) with $\mathbf{z}_n = \Phi(\mathbf{x}_n)$. After we do this, the dual

optimization problem becomes:

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^N}{\text{minimize:}} \quad & \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{z}_n^{\mathrm{T}}\mathbf{z}_m - \sum_{n=1}^{N}\alpha_n \\ \text{subject to:} \quad & \sum_{n=1}^{N} y_n\alpha_n = 0 \\ & \alpha_n \geq 0 \qquad\qquad (n = 1, \cdots, N). \end{aligned} \tag{8.29}$$

Now, recall the steps to obtain our final hypothesis. We solve the dual to obtain the optimal $\boldsymbol{\alpha}^*$. For a support vector $(\mathbf{z}_s, y_s)$ with $\alpha_s^* > 0$, define $b^* = y_s - \sum_{\alpha_n^*>0} y_n\alpha_n^*\mathbf{z}_n^{\mathrm{T}}\mathbf{z}_s$. Then, the final hypothesis from (8.25) is

$$g(\mathbf{x}) = \text{sign}\left(\sum_{n=1}^{N} y_n\alpha_n^*\mathbf{z}_n^{\mathrm{T}}\mathbf{z} + b^*\right),$$

where $\mathbf{z} = \Phi(\mathbf{x})$. In the entire dual formulation, $\mathbf{z}_n$ and $\mathbf{z}$ only appear as inner products. If we use the kernel trick to replace every inner product with the kernel function, then the *entire* process of solving the dual to getting the final hypothesis will be in terms of the kernel. We need never visit the $\mathcal{Z}$ space to explicitly construct $\mathbf{z}_n$ or $\mathbf{z}$. The algorithm box below summarizes these steps.

Hard-Margin SVM with Kernel

1: Construct $\mathrm{Q_D}$ from the kernel K, and $\mathrm{A_D}$:

$$\mathrm{Q_D} = \begin{bmatrix} y_1y_1\mathrm{K}_{11} & \dots & y_1y_N\mathrm{K}_{1N} \\ y_2y_1\mathrm{K}_{21} & \dots & y_2y_N\mathrm{K}_{2N} \\ \vdots & \vdots & \vdots \\ y_Ny_1\mathrm{K}_{N1} & \dots & y_Ny_N\mathrm{K}_{NN} \end{bmatrix} \quad \text{and} \quad \mathrm{A_D} = \begin{bmatrix} \mathbf{y}^{\mathrm{T}} \\ -\mathbf{y}^{\mathrm{T}} \\ \mathrm{I}_{N\times N} \end{bmatrix},$$

where $\mathrm{K}_{mn} = K(\mathbf{x}_m, \mathbf{x}_n)$. (K is called the *Gram* matrix.)

2: Use a QP-solver to optimize the dual problem:

$$\boldsymbol{\alpha}^* \leftarrow \mathsf{QP}(\mathrm{Q_D}, -\mathbf{1}_N, \mathrm{A_D}, \mathbf{0}_{N+2}).$$

3: Let s be any support vector for which $\alpha_s^* > 0$. Compute

$$b^* = y_s - \sum_{\alpha_n^*>0} y_n\alpha_n^* K(\mathbf{x}_n, \mathbf{x}_s).$$

4: Return the final hypothesis

$$g(\mathbf{x}) = \text{sign}\left(\sum_{\alpha_n^*>0} y_n\alpha_n^* K(\mathbf{x}_n, \mathbf{x}) + b^*\right).$$

In the algorithm, the dimension of the $\mathcal{Z}$ space has disappeared, and the running time depends on the efficiency of the kernel, and not $\tilde{d}$. For our polynomial kernel this means the efficiency is determined by d.

An efficient kernel function relies on a carefully constructed *specific* transform to allow fast computation. If we consider another 2nd-order transform

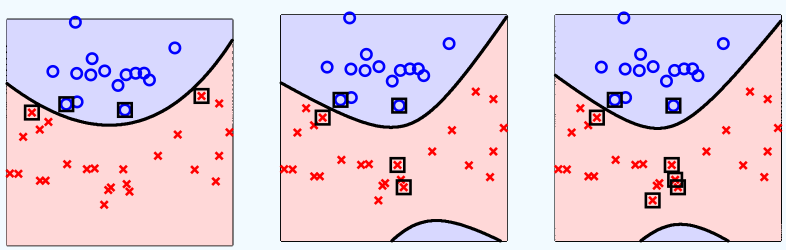
$$\Phi(\mathbf{x}) = (3, 1\cdot x_1, 4\cdot x_2, \ldots, 1\cdot x_d, 5\cdot x_1x_1, 9\cdot x_1x_2, \ldots, 2\cdot x_dx_d)$$

that multiplies each component of the original transform by some arbitrary coefficient, it would be difficult to derive an efficient kernel, even though our coefficients are related to the magic number π ☺. However, one can have certain combinations of coefficients that would still make K easy to compute. For instance, set parameters $\gamma > 0$ and $\zeta > 0$, and consider the transform

$$\Phi(\mathbf{x}) = \Big(\zeta\cdot 1, \sqrt{2\gamma\zeta}\cdot x_1, \sqrt{2\gamma\zeta}\cdot x_2, \ldots, \sqrt{2\gamma\zeta}\cdot x_d, \\ \gamma\cdot x_1x_1, \gamma\cdot x_1x_2, \ldots, \gamma\cdot x_dx_d\Big),$$

then $K(\mathbf{x}, \mathbf{x}') = (\zeta + \gamma\mathbf{x}^{\mathrm{T}}\mathbf{x}')^2$, which is also easy to compute. The resulting kernel K is often called the second-order polynomial kernel.

At first glance, the freedom to choose (γ, ζ) and still get an efficiently-computable kernel looks useful and also harmless. Multiplying each feature in the $\mathcal{Z}$ space by different constants does not change the expressive power of linear classifiers in the $\mathcal{Z}$ space. Nevertheless, changing (γ, ζ) changes the *geometry* in the $\mathcal{Z}$ space, which affects distances and hence the *margin* of a hyperplane. Thus, different (γ, ζ) could result in a different optimal hyperplane in the $\mathcal{Z}$ space since the margins of all the hyperplanes have changed, and this may give a different quadratic separator in the $\mathcal{X}$ space. The following figures show what happens on some artificial data when you vary γ with ζ fixed at 1.



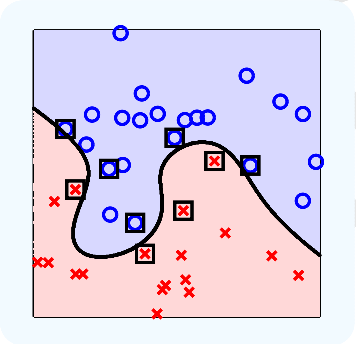
$(1 + 0.001\mathbf{x}^{\mathrm{T}}\mathbf{x}')^2$ $1 + (\mathbf{x}^{\mathrm{T}}\mathbf{x}') + (\mathbf{x}^{\mathrm{T}}\mathbf{x}')^2$ $(1 + 1000\mathbf{x}^{\mathrm{T}}\mathbf{x}')^2$

We see that the three quadratic curves are different, and so are their support vectors denoted by the squares. It is difficult, even after some 'forbidden' visual snooping ☺, to decide which curve is better. One possibility is to use the E_{cv} bound based on the number of support vectors – but keep in mind that the E_{cv} bound can be rather loose in practice. Other possibilities

include using the other validation tools that we have introduced in Chapter 4 to choose γ or other parameters in the kernel function. The choice of kernels and kernel parameters is quite important for ensuring a good performance of nonlinear SVM. Some simple guidelines for popular kernels will be discussed in Section 8.3.2.

The derivation of the second-order polynomial kernel above can be extended to the popular *degree-Q polynomial kernel*

$$K(\mathbf{x}, \mathbf{x}') = (\zeta + \gamma \mathbf{x}^{\mathrm{T}}\mathbf{x}')^{Q},$$



where $\gamma > 0$, $\zeta > 0$, and $Q \in \mathbb{N}$.[5] Then, with the kernel trick, hard-margin SVM can learn sophisticated polynomial boundaries of different orders by using exactly the same procedure and just plugging in different polynomial kernels. So, we can efficiently use very high-dimensional kernels while at the same time implicitly control the model complexity by maximizing the margin. The side figure shows a 10-th order polynomial found by kernel-SVM with margin 0.1.

Gaussian-RBF Kernel. Another popular kernel is called the Gaussian-RBF kernel,[6] which has the form

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

for some $\gamma > 0$. Let us take $\gamma = 1$ and take $\mathbf{x}$ to be a scalar $x \in \mathbb{R}$ in order to understand the transform Φ implied by the kernel. In this case,

$$\begin{aligned} K(x, x') &= \exp\left(-\|x - x'\|^2\right) \\ &= \exp\left(-(x)^2\right) \cdot \exp(2xx') \cdot \exp\left(-(x')^2\right) \\ &= \exp\left(-(x)^2\right) \cdot \left(\sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!}\right) \cdot \exp\left(-(x')^2\right), \end{aligned}$$

which is equivalent to an inner product in a feature space defined by the nonlinear transform

$$\Phi(x) = \exp(-x^2) \cdot \left(1, \sqrt{\frac{2^1}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \sqrt{\frac{2^3}{3!}}x^3, \ldots\right).$$

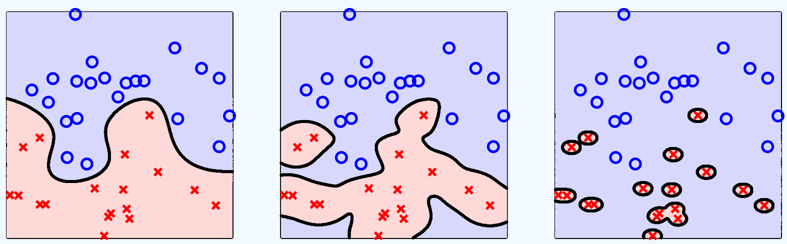
We got this nonlinear transform by splitting each term in the Taylor series of $K(x, x')$ into identical terms involving x and x'. Note that in this case, Φ is

[5] We stick to the notation Q for the order of a polynomial, not to be confused with the matrix Q in quadratic programming.

[6] RBF comes from the radial basis function introduced in Chapter 6. We use the parameter γ in place of the scale parameter $1/r$, which is common in the context of SVM.

an *infinite-dimensional transform*. Thus, a direct computation of $\Phi(x)^{\mathrm{T}}\Phi(x')$ is not possible in this case. Nevertheless, with the kernel trick, hard-margin SVM can find a hyperplane in the infinite dimensional $\mathcal{Z}$ space with model complexity under control if the margin is large enough.

The parameter γ controls the the width of the Gaussian kernel. Different choices for the width γ correspond to different geometries in the infinite-dimensional $\mathcal{Z}$ space, much like how different choices for (γ, ζ) in the polynomial kernel correspond to different geometries in the polynomial $\mathcal{Z}$ space. The following figures show the classification results of three Gaussian-RBF kernels only differing in their choice of γ.



$\exp(-1\|\mathbf{x}-\mathbf{x}'\|^2)$ $\exp(-10\|\mathbf{x}-\mathbf{x}'\|^2)$ $\exp(-100\|\mathbf{x}-\mathbf{x}'\|^2)$

When the Gaussian functions are narrow (large γ), we clearly see that even the protection of a large margin cannot suppress overfitting. However, for a reasonably small γ, the sophisticated boundary discovered by SVM with the Gaussian-RBF kernel looks quite good. Again, this demonstrates that kernels and kernel parameters need to be carefully chosen to get reasonable performance.

Exercise 8.15

Consider two finite-dimensional feature transforms Φ_1 and Φ_2 and their corresponding kernels K_1 and K_2.

(a) Define $\Phi(\mathbf{x}) = (\Phi_1(\mathbf{x}), \Phi_2(\mathbf{x}))$. Express the corresponding kernel of Φ in terms of K_1 and K_2.

(b) Consider the matrix $\Phi_1(\mathbf{x})\Phi_2(\mathbf{x})^{\mathrm{T}}$ and let $\Phi(\mathbf{x})$ be the vector representation of the matrix (say, by concatenating all the rows). Express the corresponding kernel of Φ in terms of K_1 and K_2.

(c) Hence, show that if K_1 and K_2 are kernels, then so are $K_1 + K_2$ and K_1K_2.

The results above can be used to construct the general polynomial kernels and (when extended to the infinite-dimensional transforms) to construct the general Gaussian-RBF kernels.

8.3.2 Choice of Kernels

Three kernels are popular in practice: linear, polynomial and Gaussian-RBF. The linear kernel, which corresponds to a special polynomial kernel with $Q = 1$, $\gamma = 1$, and $\zeta = 0$, corresponds to the identity transform. Solving the SVM dual problem (8.22) with the linear kernel is equivalent to solving the linear hard-margin SVM (8.4). Many special SVM packages utilize the equivalence to find the optimal $(b^*, \mathbf{w}^*)$ or $\boldsymbol{\alpha}^*$ more efficiently. One particular advantage of the linear hard-margin SVM is that the value of w_i^* can carry some explanation on how the prediction $g(\mathbf{x})$ is made, just like our familiar perceptron. One particular disadvantage of linear hard-margin SVM is the inability to produce a sophisticated boundary, which may be needed if the data is not linearly separable.

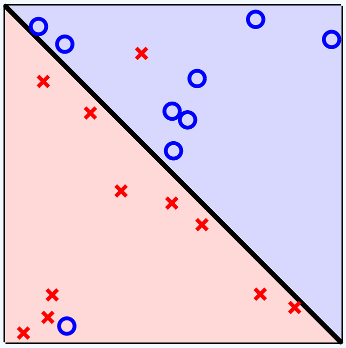
The polynomial kernel provides two controls of complexity: an explicit control from the degree Q and an implicit control from the large-margin concept. The kernel can perform well with a suitable choice of (γ, ζ) and Q. Nevertheless, choosing a good combination of three parameters is not an easy task. In addition, when Q is large, the polynomial kernel evaluates to a value with either very big or very small magnitude. The large range of values introduces numerical difficulty when solving the dual problem. Thus, the polynomial kernel is typically used only with degree $Q \le 10$, and even then, only when $\zeta + \gamma\mathbf{x}^{\mathsf{T}}\mathbf{x}'$ are scaled to reasonable values by appropriately choosing (ζ, γ).

The Gaussian-RBF kernel can lead to a sophisticated boundary for SVM while controlling the model complexity using the large-margin concept. One only needs to specify one parameter, the width γ, and its numerical range is universally chosen in the interval $[0, 1]$. This often makes it preferable to the polynomial and linear kernels. On the down side, because the corresponding transform of the Gaussian-RBF kernel is an infinite-dimensional one, the resulting hypothesis can only be expressed by the support vectors rather than the actual hyperplane, making it difficult to interpret the prediction $g(\mathbf{x})$. Interestingly, when coupling SVM with the Gaussian-RBF kernel, the hypothesis contains a linear combination of Gaussian functions centered at $\mathbf{x}_n$, which can be viewed as a special case of the RBF Network introduced in e-Chapter 6.

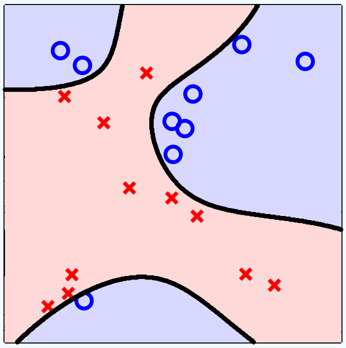
In addition to the above three kernels, there are tons of other kernel choices. One can even design new kernels that better suit the learning task at hand. Note, however, that the kernel is defined as a shortcut function for computing inner products. Thus, similar to what's shown in Exercise 8.11, the *Gram* matrix defined by

$$\mathrm{K} \;=\; \begin{bmatrix} K(\mathbf{x}_1,\mathbf{x}_1) & K(\mathbf{x}_1,\mathbf{x}_2) & \ldots & K(\mathbf{x}_1,\mathbf{x}_N) \\ K(\mathbf{x}_2,\mathbf{x}_1) & K(\mathbf{x}_2,\mathbf{x}_2) & \ldots & K(\mathbf{x}_2,\mathbf{x}_N) \\ \ldots & \ldots & \ldots & \ldots \\ K(\mathbf{x}_N,\mathbf{x}_1) & K(\mathbf{x}_N,\mathbf{x}_2) & \ldots & K(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix}$$

must be positive semi-definite. It turns out that this condition is not only necessary but also sufficient for K to be a valid kernel function that corresponds



(a) linear classifier



(b) Gaussian-RBF kernel

Figure 8.9: For (a) a noisy data set that linear classifier appears to work quite well, (b) using the Gaussian-RBF kernel with the hard-margin SVM leads to overfitting.

to the inner product in some nonlinear $\mathcal{Z}$ space. This requirement is formally known as Mercer's condition.

> $K(\mathbf{x}, \mathbf{x}')$ is a valid kernel function if and only if the kernel matrix K is always symmetric PSD for any given $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$.

The condition is often used to rule out invalid kernel functions. On the other hand, proving a kernel function to be a valid one is a non-trivial task, even when using Mercer's condition.

8.4 Soft-margin SVM

The hard-margin SVM assumes that the data is separable in the $\mathcal{Z}$ space. When we transform $\mathbf{x}$ to a high-dimensional $\mathcal{Z}$ space, or an infinite-dimensional one using the Gaussian-RBF kernel for instance, we can easily overfit the data.[7] Figure 8.9(b) depicts this situation. The hard-margin SVM coupled with the Gaussian-RBF kernel insists on fitting the two 'outliers' that are misclassified by the linear classifier, and results in an unnecessarily complicated decision boundary that should be ringing your 'overfitting alarm bells'.

For the data in Figure 8.9(b), we should use a simple linear hyperplane rather than the complex Gaussian-RBF separator. This means that we will need to accept some errors, and the hard-margin SVM cannot accommodate

[7] It can be shown that the Gaussian-RBF kernel can separate any data set (with no repeated points $\mathbf{x}_n$).

that since it is designed for perfect classification of separable data. One remedy is to consider a 'soft' formulation: try to get a large-margin hyperplane, but allow small violation of the margins or even some classification errors. As we will see, this formulation controls the SVM model's sensitivity to outliers.

The most common formulation of the (linear) *soft-margin SVM* is as follows. Introduce an 'amount' of margin violation $\xi_n \geq 0$ for each data point $(\mathbf{x}_n, y_n)$ and require that

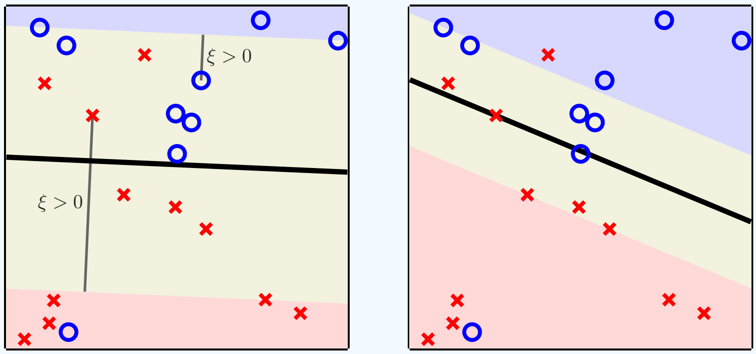
$$y_n\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b\right) \geq 1 - \xi_n.$$

According to our definition of separation in (8.2), $(\mathbf{x}_n, y_n)$ is separated if $y_n\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b\right) \geq 1$, so ξ_n captures by how much $(\mathbf{x}_n, y_n)$ fails to be separated. In terms of margin, recall that if $y_n\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b\right) = 1$ in the hard-margin SVM, then the data point is on the boundary of the margin. So, ξ_n captures how far into the margin the data point can go. The margin is not hard anymore, it is 'soft', allowing a data point to penetrate it. Note that if a point $(\mathbf{x}_n, y_n)$ satisfies $y_n\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b\right) > 1$, then the margin is not violated and the corresponding ξ_n is defined to be zero. Ideally, we would like the total sum of margin violations to be small, so we modify the hard-margin SVM to the soft-margin SVM by allowing margin violations but adding a penalty term to discourage large violations. The result is the soft-margin optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n \qquad (8.30) \\ \text{subject to} \quad & y_n\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b\right) \geq 1 - \xi_n \text{ for } n = 1, 2, \ldots, N; \\ & \xi_n \geq 0 \text{ for } n = 1, 2, \ldots, N. \end{aligned}$$

We solve this optimization problem to obtain $(b, \mathbf{w})$. Compared with (8.4), the new terms are highlighted in red. We get a large margin by minimizing the term $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$ in the objective function. We get small violations by minimizing the term $\sum_{n=1}^{N}\xi_n$. By minimizing the sum of these two terms, we get a compromise between our two goals, and that compromise will favor one or the other of our goals (large margin versus small margin violations) depending on the user-defined penalty parameter denoted by C. When C is large, it means we care more about violating the margin, which gets us closer to the hard-margin SVM. When C is small, on the other hand, we care less about violating the margin. By choosing C appropriately, we get a large-margin hyperplane (small effective d_{VC}) with a small amount of margin violations.

The new optimization problem in (8.30) looks more complex than (8.4). Don't panic. We can solve this problem using quadratic programming, just as we did with the hard-margin SVM (see Exercise 8.16 for the explicit solution). Figure 8.10 shows the result of solving (8.30) using the data from Figure 8.6(a). When C is small, we obtain a classifier with very large margin but with many cases of margin violations (some cross the boundary and some don't); when



(a) $C = 1$ (b) $C = 500$

Figure 8.10: The linear soft-margin optimal hyperplane algorithm from Exercise 8.16 on the non-separable data in Figure 8.6(a).

C is large, we obtain a classifier with smaller margin but with less margin violation.

Exercise 8.16

Show that the optimization problem in (8.30) is a QP-problem.

(a) Show that the optimization variable is $\mathbf{u} = \begin{bmatrix} b \\ \mathbf{w} \\ \boldsymbol{\xi} \end{bmatrix}$, where $\boldsymbol{\xi} = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_N \end{bmatrix}$.

(b) Show that $\mathbf{u}^* \leftarrow \text{QP}(\text{Q}, \mathbf{p}, \text{A}, \mathbf{c})$, where

$$\text{Q} = \begin{bmatrix} 0 & \mathbf{0}_d^{\text{T}} & \mathbf{0}_N^{\text{T}} \\ \mathbf{0}_d & \text{I}_d & \mathbf{0}_{d\times N} \\ \mathbf{0}_N & \mathbf{0}_{N\times d} & \mathbf{0}_{N\times N} \end{bmatrix}, \ \mathbf{p} = \begin{bmatrix} \mathbf{0}_{d+1} \\ C\cdot\mathbf{1}_N \end{bmatrix}, \ \text{A} = \begin{bmatrix} \text{YX} & \text{I}_N \\ \mathbf{0}_{N\times(d+1)} & \text{I}_N \end{bmatrix}, \ \mathbf{c} = \begin{bmatrix} \mathbf{1}_N \\ \mathbf{0}_N \end{bmatrix},$$

and YX is the signed data matrix from Exercise 8.4.

(c) How do you recover b^*, $\mathbf{w}^*$ and $\boldsymbol{\xi}^*$ from $\mathbf{u}^*$?

(d) How do you determine which data points violate the margin, which data points are on the edge of the margin and which data points are correctly separated and outside the margin?

Similar to the hard-margin optimization problem (8.4), the soft-margin version (8.30) is a convex QP-problem. The corresponding dual problem can be derived using the same technique introduced in Section 8.2. We will just provide the very first step here, and let you finish the rest. For the soft-margin

SVM (8.30), the Lagrange function is

$$\begin{aligned}&\mathcal{L}(b, \mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})\\ =\ & \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n(1-\xi_n - y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)) - \sum_{n=1}^{N}\beta_n\xi_n,\end{aligned}$$

where $\alpha_n \geq 0$ are the Lagrange multipliers for $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 - \xi_n$, and $\beta_n \geq 0$ are the Lagrange multipliers for $\xi_n \geq 0$. Then, the KKT condition tells us that at the optimal solution, $\partial\mathcal{L}/\partial\xi_n$ has to be 0. This means

$$C - \alpha_n - \beta_n = 0.$$

That is, α_n and β_n sum to C. We can then replace all β_n by $C - \alpha_n$ without loss of optimality. If we do so, the Lagrange dual problem simplifies to

$$\max_{\substack{\boldsymbol{\alpha}\geq\mathbf{0},\boldsymbol{\beta}\geq\mathbf{0},\\ \alpha_n+\beta_n=C}} \ \min_{b,\mathbf{w},\boldsymbol{\xi}} \quad \mathcal{L}(b, \mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}),$$

where

$$\begin{aligned}\mathcal{L} &= \tfrac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n(1-\xi_n - y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)) - \sum_{n=1}^{N}(C-\alpha_n)\xi_n\\ &= \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \sum_{n=1}^{N}\alpha_n(1 - y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b)).\end{aligned}$$

Does the simplified objective function now look familiar? It is just the objective function that led us to the dual of the hard-margin SVM! ☺ We trust that you can then go through all the steps as in Section 8.2 to get the whole dual. When expressed in matrix form, the dual problem of (8.30) is

$$\begin{aligned}\min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathrm{Q_D}\boldsymbol{\alpha} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\alpha}\\ \text{subject to} \quad & \mathbf{y}^{\mathrm{T}}\boldsymbol{\alpha} = 0;.\\ & \mathbf{0} \leq \boldsymbol{\alpha} \leq C\cdot\mathbf{1}.\end{aligned}$$

Interestingly, compared with (8.22), the only change of the dual problem is that each α_n is now upper-bounded by C, the penalty rate, instead of ∞. The formulation can again be solved by some general or specifically tailored quadratic programming packages.

For the soft-margin SVM, we get a similar expression for the final hypothesis in terms of the optimal dual solution, like that of the hard-margin SVM.

$$g(\mathbf{x}) = \text{sign}\left(\sum_{\alpha_n^*>0} y_n\alpha_n^*\mathbf{x}_n^{\mathrm{T}}\mathbf{x} + b^*\right).$$

The kernel trick is still applicable as long as we can calculate the optimal b^* efficiently. Getting the optimal b^* from the optimal $\boldsymbol{\alpha}^*$, however, is slightly more complicated in the soft-margin case. The KKT conditions state that

$$\begin{aligned} \alpha_n^* \cdot \left(y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) - 1 + \xi_n^* \right) &= 0. \\ \beta_n^* \cdot \xi_n^* = (C - \alpha_n^*) \cdot \xi_n^* &= 0. \end{aligned}$$

If $\alpha_n^* > 0$, then $\left(y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) - 1 + \xi_n^* \right) = 0$ and hence

$$y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) = 1 - \xi_n^* \leq 1.$$

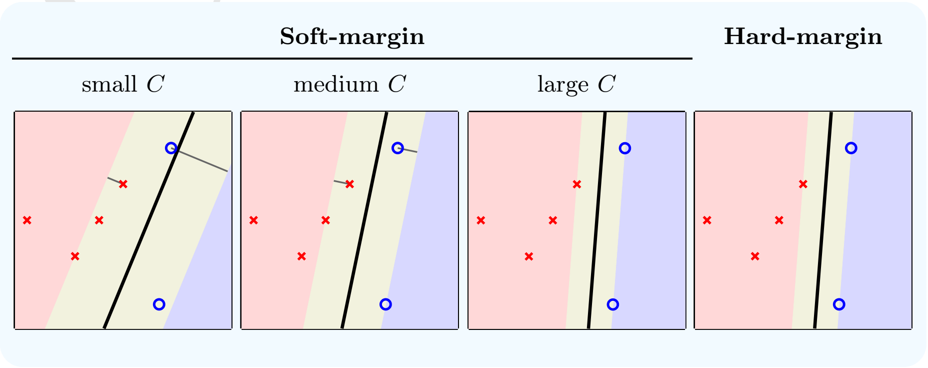
On the other hand, if $\alpha_n^* < C$, then $\xi_n^* = 0$ and hence

$$y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) \geq 1.$$

The two inequalities gives a range for the optimal b^*. When there is a support vector with $0 < \alpha_n^* < C$, we see that the inequalities can be combined to an equality $y_n(\mathbf{w}^{*\mathrm{T}}\mathbf{x}_n + b^*) = 1$, which can be used to pin down the optimal b^* as we did for the hard-margin SVM. In other cases, there are many choices of b^* and you can freely pick any.

The support vectors with $0 < \alpha_n^* < C$ are called the free support vectors, which are guaranteed to be on the boundary of the fat-hyperplane and hence also called margin support vectors. The support vectors with $\alpha_n^* = C$, on the other hand, are called the bounded support vectors (also called non-margin support vectors). They can be on the fat boundary, slightly violating the boundary but still correctly predicted, or seriously violating the boundary and erroneously predicted.

For separable data (in the Φ-transformed space), there exists some C' such that whenever $C \geq C'$, the soft-margin SVM produces exactly the same solution as the hard-margin SVM. Thus, the hard-margin SVM can be viewed as a special case of the soft-margin one, illustrated below.



Let's take a closer look at the parameter C in the soft-margin SVM formulation (8.30). If C is large, it means that we do want all violations (errors) ξ_n to be as small as possible with the possible trade-off being a smaller margin (higher complexity). If C is small, we will tolerate some amounts of errors, while possibly getting a less complicated hypothesis with large margin. Does the trade-off sound familiar? We have encountered such a trade-off in Chapter 4 when we studied regularization. Let

$$E_{\text{SVM}}(b, \mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \max(1 - y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b), 0).$$

The n-th term in $E_{\text{SVM}}(b, \mathbf{w})$ evaluates to 0 if there is no violation from the n-th example, so $y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1$; otherwise, the nth term is the amount of violation for the corresponding data point. Therefore, the objective that we minimize in soft-margin SVM (8.30) can be re-written as the following optimization problem

$$\min_{b,\mathbf{w}} \lambda\mathbf{w}^{\mathrm{T}}\mathbf{w} + E_{\text{SVM}}(b, \mathbf{w}),$$

subject to the constraints, and where $\lambda = 1/2CN$. In other words, soft-margin SVM can be viewed as a special case of regularized classification with $E_{\text{SVM}}(b, \mathbf{w})$ as a surrogate for the in-sample error and $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$ (without b) as the regularizer. The $E_{\text{SVM}}(b, \mathbf{w})$ term is an upper bound on the classification in-sample error E_{in}, while the regularizer term comes from the large-margin concept and controls the effective model complexity.

Exercise 8.17

Show that $E_{\text{SVM}}(b, \mathbf{w})$ is an upper bound on the $E_{\text{in}}(b, \mathbf{w})$, where E_{in} is the classification 0/1 error.

In summary, soft-margin SVM can:

1. Deliver a large-margin hyperplane, and in so doing it can control the effective model complexity.
2. Deal with high- or infinite-dimensional transforms using the kernel trick,
3. Express the final hypothesis $g(\mathbf{x})$ using only a few support vectors, their corresponding Lagrange multipliers, and the kernel.
4. Control the sensitivity to outliers and regularize the solution through setting C appropriately.

When the regularization parameter C and the kernel are chosen properly, the soft-margin SVM is often observed to enjoy a low E_{out} with the useful properties above. These properties make the soft-margin SVM (*the* SVM for short) one of the most useful classification models and often the first choice in learning from data. It is a robust linear model with advanced nonlinear transform capability when used with a kernel.

8.5 Problems

Problem 8.1 Consider a data set with two data points $\mathbf{x}_\pm \in \mathbb{R}^d$ having class ± 1 respectively. Manually solve (8.4) by explicitly minimizing $\|\mathbf{w}\|^2$ subject to the two separation constraints.

Compute the optimal (maximum margin) hyperplane $(b^*, \mathbf{w}^*)$ and its margin. Compare with your solution to Exercise 8.1.

Problem 8.2 Consider a data set with three data points in $\mathbb{R}^2$:

$$\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 0 & -1 \\ -2 & 0 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}$$

Manually solve (8.4) to get the optimal hyperplane $(b^*, \mathbf{w}^*)$ and its margin.

Problem 8.3 Manually solve the dual optimization from Example 8.8 to obtain the same $\boldsymbol{\alpha}^*$ that was obtained in the text using a QP-solver. Use the following steps.

(a) Show that the dual optimization problem is to minimize

$$\mathcal{L}(\boldsymbol{\alpha}) = 4\alpha_2^2 + 2\alpha_3^2 + \tfrac{9}{2}\alpha_4^2 - 4\alpha_2\alpha_3 - 6\alpha_2\alpha_4 + 6\alpha_3\alpha_4 - \alpha_1 - \alpha_2 - \alpha_3 - \alpha_4,$$

subject to the constraints

$$\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4;$$
$$\alpha_1, \alpha_2, \alpha_3, \alpha_4 \geq 0.$$

(b) Use the equality constraint to replace α_1 in $\mathcal{L}(\boldsymbol{\alpha})$ to get

$$\mathcal{L}(\boldsymbol{\alpha}) = 4\alpha_2^2 + 2\alpha_3^2 + \tfrac{9}{2}\alpha_4^2 - 4\alpha_2\alpha_3 - 6\alpha_2\alpha_4 + 6\alpha_3\alpha_4 - 2\alpha_3 - 2\alpha_4.$$

(c) Fix $\alpha_3, \alpha_4 \geq 0$ and minimize $\mathcal{L}(\boldsymbol{\alpha})$ in (b) with respect to α_2 to show that

$$\alpha_2 = \frac{\alpha_3}{2} + \frac{3\alpha_4}{4} \qquad \text{and} \qquad \alpha_1 = \alpha_3 + \alpha_4 - \alpha_2 = \frac{\alpha_3}{2} + \frac{\alpha_4}{4}.$$

Are these valid solutions for α_1, α_2?

(d) Use the expressions in (c) to reduce the problem to minimizing

$$\mathcal{L}(\boldsymbol{\alpha}) = \alpha_3^2 + \tfrac{9}{4}\alpha_4^2 + 3\alpha_3\alpha_4 - 2\alpha_3 - 2\alpha_4,$$

subject to $\alpha_3, \alpha_4 \geq 0$. Show that the minimum is attained when $\alpha_3 = 1$ and $\alpha_4 = 0$. What are α_1, α_2?

It's a relief to have QP-solvers for solving such problems in the general case!

Problem 8.4 Set up the dual problem for the toy data set in Exercise 8.2. Then, solve the dual problem and compute $\boldsymbol{\alpha}^*$, the optimal Lagrange multipliers.

Problem 8.5 [Bias and Variance of the Optimal Hyperplane] In this problem, you are to investigate the bias and variance of the optimal hyperplane in a simple setting. The input is $(x_1, x_2) \in [-1, 1]^2$ and the target function is $f(\mathbf{x}) = \text{sign}(x_2)$.

The hypothesis set $\mathcal{H}$ contains horizontal linear separators $h(\mathbf{x}) = \text{sign}(x_2 - a)$, where $-1 \le a \le 1$. Consider two algorithms:

Random: Pick a random separator from $\mathcal{H}$.

SVM: Pick the maximum margin separator from $\mathcal{H}$.

(a) Generate 3 data point uniformly in the upper half of the input-space and 3 data points in the lower half, and obtain $g_{\textsf{Random}}$ and $g_{\textsf{SVM}}$.

(b) Create a plot of your data, and your two hypotheses.

(c) Repeat part (a) for a million data sets to obtain one million Random and SVM hypotheses.

(d) Give a histogram of the values of $a_{\textsf{Random}}$ resulting from the random algorithm and another histogram of $a_{\textsf{SVM}}$ resulting from the optimal separators. Compare the two histograms and explain the differences.

(e) Estimate the bias and var for the two algorithms. Explain your findings, in particular which algorithm is better for this toy problem.

Problem 8.6 Show that $\sum\limits_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}\|^2$ is minimized at $\boldsymbol{\mu} = \frac{1}{N}\sum\limits_{n=1}^{N} \mathbf{x}_n$.

Problem 8.7 For any $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with $\|\mathbf{x}_n\| \le R$ and N even, show that there exists a balanced dichotomy $y_1, \ldots, y_n$ that satisfies

$$\sum_{n=1}^{N} y_n = 0, \text{ and } \qquad \left\| \sum_{n=1}^{N} y_n \mathbf{x}_n \right\| \le \frac{NR}{\sqrt{N-1}}.$$

(This is the geometric lemma that is need to bound the VC-dimension of ρ-fat hyperplanes by $\lceil R^2/\rho^2 \rceil + 1$.) The following steps are a guide for the proof.

Suppose you randomly select $N/2$ of the labels $y_1, \ldots, y_N$ to be $+1$, the others being -1. By construction, $\sum_{n=1}^{N} y_n = 0$.

(a) Show $\left\| \sum\limits_{n=1}^{N} y_n \mathbf{x}_n \right\|^2 = \sum\limits_{n=1}^{N} \sum\limits_{m=1}^{N} y_n y_m \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m$.

(b) When $n = m$, what is $y_n y_m$? Show that $\mathbb{P}[y_n y_m = 1] = (\frac{N}{2} - 1)/(N - 1)$ when $n \neq m$. Hence show that

$$\mathbb{E}[y_n y_m] = \begin{cases} 1 & m = n; \\ -\frac{1}{N-1} & m \neq n. \end{cases}$$

(c) Show that

$$\mathbb{E}\left[\left\|\sum_{n=1}^{N} y_n \mathbf{x}_n\right\|^2\right] = \frac{N}{N-1} \sum_{n=1}^{N} \|\mathbf{x}_n - \bar{\mathbf{x}}\|^2,$$

where the average vector $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$. *[Hint: Use linearity of expectation in (a), and consider the cases $m = n$ and $m \neq n$ separately.]*

(d) Show that $\sum_{n=1}^{N} \|\mathbf{x}_n - \bar{\mathbf{x}}\|^2 \leq \sum_{n=1}^{N} \|\mathbf{x}_n\|^2 \leq NR^2$ *[Hint: Problem 8.6.]*

(e) Conclude that

$$\mathbb{E}\left[\left\|\sum_{n=1}^{N} y_n \mathbf{x}_n\right\|^2\right] \leq \frac{N^2 R^2}{N-1},$$

and hence that

$$\mathbb{P}\left[\left\|\sum_{n=1}^{N} y_n \mathbf{x}_n\right\| \leq \frac{NR}{\sqrt{N-1}}\right] > 0.$$

This means for some choice of y_n, $\left\|\sum_{n=1}^{N} y_n \mathbf{x}_n\right\| \leq NR/\sqrt{N-1}$

This proof is called a probabilistic existence proof: if some random process can generate an object with *positive probability*, then that object must exist. Note that you prove existence of the required dichotomy without actually constructing it. In this case, the easiest way to construct a desired dichotomy is to randomly generate the balanced dichotomies until you have one that works.

Problem 8.8 We showed that if N points in the ball of radius R are shattered by hyperplanes with margin ρ, then $N \leq R^2/\rho^2 + 1$ when N is even. Now consider N odd, and $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with $\|\mathbf{x}_n\| \leq R$ shattered by hyperplanes with margin ρ. Recall that $(\mathbf{w}, b)$ implements $y_1, \ldots, y_N$ with margin ρ if

$$\rho\|\mathbf{w}\| \leq y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b), \qquad \text{for } n = 1, \ldots, N. \tag{8.31}$$

Show that for $N = 2k + 1$ (odd), $N \leq R^2/\rho^2 + \frac{1}{N} + 1$ as follows:

Consider random labelings $y_1, \ldots, y_N$ of the N points in which k of the labels are $+1$ and $k+1$ are -1. Define $\ell_n = \frac{1}{k}$ if $y_n = +1$ and $\ell_n = \frac{1}{k+1}$ if $y_n = -1$.

(a) For any labeling with k labels being $+1$, show, by summing (8.31) and

using the Cauchy-Schwarz inequality, that

$$2\rho \le \left\| \sum_{n=1}^{N} \ell_n y_n \mathbf{x}_n \right\|.$$

(b) Show that there exists a labeling, with k labels being $+1$, for which

$$\left\| \sum_{n=1}^{N} \ell_n y_n \mathbf{x}_n \right\| \le \frac{2NR}{(N-1)\sqrt{N+1}}$$

.

(i) Show $\left\| \sum_{n=1}^{N} \ell_n y_n \mathbf{x}_n \right\|^2 = \sum_{n=1}^{N} \sum_{m=1}^{N} \ell_n \ell_m y_n y_m \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m$.

(ii) For $m = n$, show $\mathbb{E}[\ell_n \ell_m y_n y_m] = \dfrac{1}{k(k+1)}$.

(iii) For $m \ne n$, show $\mathbb{E}[\ell_n \ell_m y_n y_m] = -\dfrac{1}{(N-1)k(k+1)}$.

[Hint: $\mathbb{P}[\ell_n \ell_m y_n y_m = 1/k^2] = k(k-1)/N(N-1)$.*]*

(iv) Show $\mathbb{E}\left[\left\| \sum_{n=1}^{N} \ell_n y_n \mathbf{x}_n \right\|^2\right] = \dfrac{N}{(N-1)k(k+1)} \sum_{n=1}^{N} \|\mathbf{x}_n - \bar{\mathbf{x}}\|^2$.

(v) Use Problem 8.6 to conclude the proof as in Problem 8.7.

(c) Use (a) and (b) to show that $N \le \dfrac{R^2}{\rho^2} + \dfrac{1}{N} + 1$.

Problem 8.9 Prove that for the separable case, if you remove a data point that is not a support vector, then the maximum margin classifier does not change. You may use the following steps are a guide. Let g be the maximum margin classifier for all the data, and g^- the maximum margin classifier after removal of a data point that is not a support vector.

(a) Show that g is a separator for $\mathcal{D}^-$, the data minus the non-support vector.

(b) Show that the maximum margin classifier is unique.

(c) Show that if g^- has larger margin than g on $\mathcal{D}^-$, then it also has larger margin on $\mathcal{D}$, a contradiction. Hence, conclude that g is the maximum margin separator for $\mathcal{D}^-$.

Problem 8.10 An essential support vector is one whose removal from the data set changes the maximum margin separator. For the separable case, show that there are at most $d+1$ essential support vectors. Hence, show that for the separable case,

$$E_{\text{cv}} \le \frac{d+1}{N}.$$

Problem 8.11 Consider the version of the PLA that uses the misclassified data point $\mathbf{x}_n$ with lowest index n for the weight update. Assume the data is separable and given in some fixed but arbitrary order, and when you remove a data point, you do not alter this order. In this problem, prove that

$$E_{\text{cv}}(\mathsf{PLA}) \leq \frac{R^2}{N\rho^2},$$

where ρ is the margin (half the width) of the maximum margin separating hyperplane that would (for example) be returned by the SVM. The following steps are a guide for the proof.

(a) Use the result in Problem 1.3 to show that the number of updates T that the PLA makes is at most
$$T \leq \frac{R^2}{\rho^2}.$$

(b) Argue that this means that PLA only 'visits' at most R^2/ρ^2 different points during the course of its iterations.

(c) Argue that after leaving out any point $(\mathbf{x}_n, y_n)$ that is not 'visited', PLA will return the same classifier.

(d) What is the leave-one-out error e_n for these points that were not visited? Hence, prove the desired bound on $E_{\text{cv}}(\mathsf{PLA})$.

Problem 8.12 Show that optimal solution for soft-margin optimal hyperplane (solving optimization problem (8.30)) with $C \to \infty$ will be the same solution that was developed using linear programming in Problem 3.6(c).

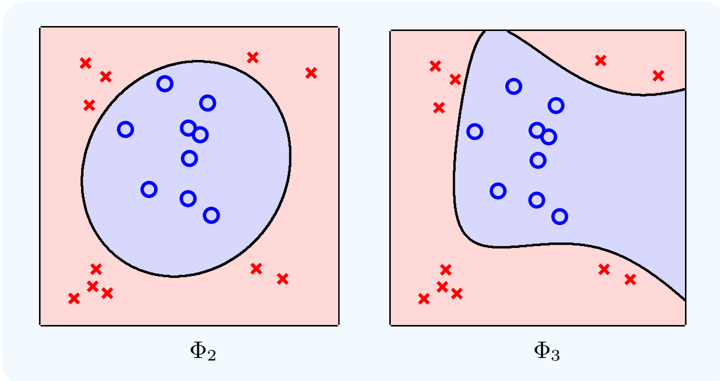
Problem 8.13 The data for Figure 8.6(b) are given below:

$y_n = +1$	$y_n = -1$
	$(0.491, 0.920)$
$(-0.494, 0.363)$	$(-0.892, -0.946)$
$(-0.311, -0.101)$	$(-0.721, -0.710)$
$(-0.0064, 0.374)$	$(0.519, -0.715)$
$(-0.0089, -0.173)$	$(-0.775, 0.551)$
$(0.0014, 0.138)$	$(-0.646, 0.773)$
$(-0.189, 0.718)$	$(-0.803, 0.878)$
$(0.085, 0.32208)$	$(0.944, 0.801)$
$(0.171, -0.302)$	$(0.724, -0.795)$
$(0.142, 0.568)$	$(-0.748, -0.853)$
	$(-0.635, -0.905)$

Use the data on the left with the 2nd and 3rd order polynomial transforms Φ_2, Φ_3 and the pseudo-inverse algorithm for linear regression from Chapter 3 to get weights $\tilde{\mathbf{w}}$ for your final final hypothesis in $\mathcal{Z}$-space. The final hypothesis in $\mathcal{X}$-space is:

$$g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^{\mathrm{T}}\Phi(\mathbf{x}) + \tilde{b}).$$

(a) Plot the classification regions for your final hypothesis in $\mathcal{X}$-space. Your results should look something like:



(b) Which of fits in part (a) appears to have overfitted?

(c) Use the pseudo-inverse algorithm with regularization parameter $\lambda = 1$ to address the overfitting you identified in part (c). Give a plot of the resulting classifier.

Problem 8.14 The kernel trick can be used with any model as long as fitting the data and the final hypothesis only require the computation of dot-products in the $\mathcal{Z}$-space. Suppose you have a kernel K, so

$$\Phi(\mathbf{x})^{\mathrm{T}}\Phi(\mathbf{x}') = K(\mathbf{x}, \mathbf{x}').$$

Let Z be the data in the $\mathcal{Z}$-space. The pseudo-inverse algorithm for regularized regression computes optimal weights $\tilde{\mathbf{w}}^*$ (in the $\mathcal{Z}$-space) that minimize

$$E_{\text{aug}}(\tilde{\mathbf{w}}) = \|\mathrm{Z}\tilde{\mathbf{w}} - \mathbf{y}\|^2 + \lambda \tilde{\mathbf{w}}^{\mathrm{T}}\tilde{\mathbf{w}}.$$

The final hypothesis is $g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^{\mathrm{T}}\Phi(\mathbf{x}))$. Using the representor theorem, the optimal solution can be written $\tilde{\mathbf{w}}^* = \sum_{n=1}^{N} \beta_n^* \mathbf{z}_n = \mathrm{Z}^{\mathrm{T}}\boldsymbol{\beta}^*$

(a) Show that $\boldsymbol{\beta}^*$ minimizes

$$E(\boldsymbol{\beta}) = \|\mathrm{K}\boldsymbol{\beta} - \mathbf{y}\|^2 + \lambda \boldsymbol{\beta}^{\mathrm{T}}\mathrm{K}\boldsymbol{\beta},$$

where K is the $N \times N$ Kernel-Gram matrix with entries $\mathrm{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.

(b) Show that K is symmetric.

(c) Show that the solution to the minimization problem in part (a) is:

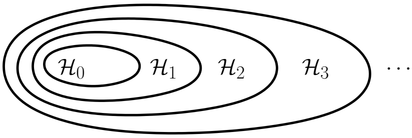
$$\boldsymbol{\beta}^* = (\mathrm{K} + \lambda \mathrm{I})^{-1}\mathbf{y}.$$

Can $\boldsymbol{\beta}^*$ be computed without ever 'visiting' the $\mathcal{Z}$-space?

(d) Show that the final hypothesis is

$$g(\mathbf{x}) = \text{sign}\left(\sum_{n=1}^{N} \beta_n^* K(\mathbf{x}_n, \mathbf{x})\right).$$

Problem 8.15 **Structural Risk Minimization (SRM).** SRM is a useful framework for model selection. A *structure* is a nested sequence of hypothesis sets:



Suppose we use in-sample error minimization as our learning algorithm in each $\mathcal{H}_m$, so $g_m = \underset{h\in\mathcal{H}_m}{\text{argmin}}\, E_{\text{in}}(h)$, and select $g^* = \underset{g_m}{\text{argmin}}\, E_{\text{in}}(g_m) + \Omega(\mathcal{H}_m)$.

(a) Show that the in-sample error $E_{\text{in}}(g_m)$ is non-increasing in m. What about the penalty $\Omega(\mathcal{H}_m)$? How do you expect the VC-bound to behave with m.

(b) Assume that $g^* \in \mathcal{H}_m$ with *a priori* probability p_m. (In general, the p_m are not known.) Since $\mathcal{H}_m \subset \mathcal{H}_{m+1}$, $p_0 \le p_1 \le p_2 \le \cdots \le 1$. What components of the learning problem do the p_m's depend on.

(c) Suppose $g^* = g_m \in \mathcal{H}_m$. Show that

$$\mathbb{P}\left[|E_{\text{in}}(g_i) - E_{\text{out}}(g_i)| > \epsilon \mid g^* = g_m\right] \le \frac{1}{p_i}\cdot 4m_{\mathcal{H}_i}(2N)e^{-\epsilon^2 N/8}.$$

Here, the conditioning is on selecting the function $g_m \in \mathcal{H}_m$. *[Hint: Bound the probability* $\mathbb{P}\left[\max_{g\in\mathcal{H}_i}|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \mid g^* = g_m\right]$. *Use Bayes theorem to rewrite this as* $\frac{1}{p_i}\mathbb{P}\left[\max_{g\in\mathcal{H}_i}|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \textbf{ and } g^* = g_m\right]$. *Use the fact that* $\mathbb{P}[A \textbf{ and } B] \le \mathbb{P}[A]$. *and argue that you can apply the familiar VC-inequality to the resulting expression.]*

You may interpret this result as follows: if you use SRM and end up with g_m, then the generalization bound is a factor $\frac{1}{p_m}$ worse than the bound you would have gotten had you simply started with $\mathcal{H}_m$; that is the price you pay for allowing yourself the possibility to search more than $\mathcal{H}_m$. Typically simpler models occur earlier in the structure, and so the bound will be reasonable if the target function is simple (in which case p_m is large for small m). SRM works well in practice.

Problem 8.16 Which can be posed within the SRM framework: selection among different soft order constraints $\{\mathcal{H}_C\}_{C>0}$ or selecting among different regularization parameters $\{\mathcal{H}_\lambda\}_{\lambda>0}$ where the hypothesis set fixed at $\mathcal{H}$ and the learning algorithm is augmented error minimization with different regularization parameters λ.

Problem 8.17 Suppose we use "SRM" to select among an arbitrary set of models $\mathcal{H}_1, \ldots, \mathcal{H}_M$ with $d_{\text{VC}}(\mathcal{H}_{m+1}) > d_{\text{VC}}(\mathcal{H}_m)$ (as opposed to a structure in which the additional condition $\mathcal{H}_m \subset \mathcal{H}_{m+1}$ holds).

(a) Is it possible for $E_{\text{in}}(\mathcal{H}_m) < E_{\text{in}}(\mathcal{H}_{m+1})$?

(b) Let p_m be the probability that the process leads to a function $g_m \in \mathcal{H}_m$, with $\sum_m p_m = 1$. Give a bound for the generalization error in terms of $d_{\text{VC}}(\mathcal{H}_m)$.

Problem 8.18 Suppose that we can order the hypotheses in a model, $\mathcal{H} = \{h_1, h_2, \ldots\}$. Assume that $d_{\text{VC}}(\mathcal{H})$ is infinite. Define the hypothesis subsets $\mathcal{H}_m = \{h_1, h_2, \ldots, h_m\}$. Suppose you implement a learning algorithm for $\mathcal{H}$ as follows: start with h_1; if $E_{\text{in}}(h_1) \le \nu$, stop and output h_1; if not try h_2; and so on ...

(a) Suppose that you output h_m, so you have effectively only searched the m hypotheses in $\mathcal{H}_m$. Can you use the VC-bound: (with high probability) $E_{\text{out}}(h_m) \le \nu + \sqrt{\frac{\ln(2m/\delta)}{2N}}$? If yes, why? If no, why not?

(b) Formulate this process within the SRM framework. *[Hint: the $\mathcal{H}_m$'s form a structure.]*

(c) Can you make *any* generalization conclusion (remember, $d_{\text{VC}}(\mathcal{H}) = \infty$)? If yes, what is the bound on the generalization error, and when do you expect good generalization? If no, why?